KOCAELİ ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ

BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ BÖLÜMÜ

PROJE A

İNSAN KAYNAKLAR İÇİN PERSONEL VERİMLİLİĞİ TAKİP SİSTEMİ

Muhammed BEDAVİ

KOCAELİ 2021

KOCAELİ ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ

BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ BÖLÜMÜ

PROJE A

İNSAN KAYNAKLAR İÇİN PERSONEL VERİMLİLİĞİ TAKİP SİSTEMİ

Muhammed BEDAVİ

Dr. Öğr. Üyesi, Serdar SOLAK
Danışman, Kocaeli Üniv. .....

.....

.....

Tezin Savunulduğu Tarih: 21.01.2021

ÖNSÖZ VE TEŞEKKÜR

Bu tez çalışması, Covid-19 pandeminde uzaktan çalışamaya başlayan butik şirketler için çalışma verimliğini artmak amacıyla gerçekleştirilmiştir.

Tez çalışmamda desteğini esirgemeyen, çalışmalarıma yön veren, bana güvenen ve yüreklendiren danışmanım Serdar SOLAK'a sonsuz teşekkürlerimi sunarım.

Hayatım boyunca bana güç veren en büyük destekçilerim, her aşamada sıkıntılarımı ve mutluluklarımı paylaşan sevgili aileme teşekkürlerimi sunarım.

Ocak– 2021 Muhammed BEDAVİ

Bu dokümandaki tüm bilgiler, etik ve akademik kurallar çerçevesinde elde edilip sunulmuştur. Ayrıca yine bu kurallar çerçevesinde kendime ait olmayan ve kendimin üretmediği ve başka kaynaklardan elde edilen bilgiler ve materyaller (text, resim, şekil, tablo vb.) gerekli şekilde referans edilmiş ve dokümanda belirtilmiştir.

Öğrenci No: 161307066

Adı Soyadı: Muhammed BEDAVİ

İmza:

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

İNSAN KAYNAKLAR İÇİN PERSONEL VERİMLİLİĞİ TAKİP SİSTEMİ

ÖZET

Günümüzdeki gelişen ve büyüyen teknoloji ile şirketler yaptıkları işlerin büyük bir kısmını dijital ortamlarında yapmaktadır. Bu işleri başarılı bir şekilde yapmak için büyük firmalarda çok gelişmiş ve her detayın hem kontrolünü hem de takibini sağlayan programlar vardır. Fakat orta veya küçük (Butik) firmalarda ya programların pahalı olması ya da yöneticiler gerek duymadığı için takip işlemleri klasik yöntemler ile yapılmaktadır.

2020'de tüm dünyayı etkisi altında alan Corona virüsü nedeniyle çoğu firmalar uzaktan veya dönüşümlü çalışma sistemlerine geçmek zorunda kalmıştır. O zamana kadar hiç personel takip sistemi kullanmayan şirketler uzaktan çalışma sistemine geçince büyük bir sıkıntıya düşmüştür.

Bu problemi çözmek için kapsamlı, yerli ve pandemiye uygun bir sistem geliştirilmiştir. Bu sistem 24 saat çalışabilen, işlemciye yükü olmayan ve offline/online moduna sahip olan bir agent yazılımı yardımıyla firma personellerin çalışma saatleri içinde yaptıkları işlemler, kullandıkları programlar, girdikleri siteler, klavye ve fare istatistikleri vb. verileri otomatik bir şekilde firma sunucuna göndermektedir. Bu verileri kullanarak hem personel takibi hem de işlemlerin verimliliğini artmak mümkündür.

Anahtar kelimeler: takip sistemi, uzaktan çalışma, veri bilimi, istatistik.

PERSONNEL PRODUCTİVİTY MONITORING SYSTEM FOR HUMAN RESOURCES

ABSTRACT

With today's developing and growing technology, companies do most of their work in digital environments. In order to do these works successfully, there are very developed programs in large companies that provide both control and follow-up of every detail. However, in medium or small (boutique) companies, follow-up processes are carried out with classical methods, either because the programs are expensive or the managers do not need them.

Due to the Corona virus, which affected the whole world in 2020, most companies had to switch to remote or rotary working systems. Until then, companies that had never used a personnel tracking system faced great difficulties when they switched to remote working systems.

A comprehensive, domestic and pandemic system has been developed to solve this problem. With the help of an agent software that can work 24 hours a day, has no load on the processor and has an offline / online mode, the company personnel's operations during working hours, the programs they use, the sites they enter, the keyboard and mouse statistics, etc. it automatically sends the data to the company server. By using these data, it is possible to increase both personnel tracking and the efficiency of operations.

Keywords: Tracking system, Work from home, Data Science, statistics.

1. PROJE AMACI, ÖNEMİ VE KAPSAMI

Günümüzdeki gelişen ve büyüyen teknoloji ile şirketler yaptıkları işlerin büyük bir kısmını dijital ortamlarında yapmaktadır. Bu işleri başarılı bir şekilde yapmak için büyük firmalarda çok gelişmiş ve her detayın hem kontrolünü hem de takibini sağlayan programlar vardır. Bu programlar büyük ölçekli şirketlere uygun bir şekilde yapıldığı için hem alım maliyeti hem de yazılımları çalıştırmak için pahalı bir donanımı gerektirir. Orta veya küçük (Butik) firmalarda ya programların pahalı olması ya da yöneticiler gerek duymadığı için takip işlemleri klasik yöntemler ile yapılmaktadır. Bu firmaların işleri çok kritik veya yoğun olmadığı için işlerini klasik yöntemler ile halledebilmekteydi.

2020'de tüm dünyayı etkisi altında alan Corona virüsü nedeniyle çoğu firmalar uzaktan veya dönüşümlü çalışma sistemlerine geçmek zorunda kalmıştır. O zamana kadar hiç personel takip sistemi kullanmayan şirketler uzaktan çalışma sistemine geçince büyük bir sıkıntıya düşmüştür. Böyle firmaların ne maddi imkanları ne de teknik deneyimleri piyasadaki büyük programları almaya yeterli değildi. Piyasadaki küçük ölçekli şirketlere yönelik personel takip ve yönetim sistemleri oldukça basit ve tek amaçlı olduğu için pandemi sürecinde yetersiz kalmıştır. Yukarıda belirtilen sebeplerden dolayı hem kapsamlı hem de butik firmalara uygun bir yazılıma ihtiyaç duyulmuştur.

1.1. Piyasa Araştırması

Tezin anlattığı yazılıma başlamadan önce piyasadaki yazılımlara dikkatli bir şekilde bakılıp hangi problemleri ve nasıl çözdüğü araştırılmıştır.

Araştırma yapıldıktan sonra yazılımların hem ortak özellikleri hem de her yazılımın ortaya çıkarmış olduğu önemli özellikleri bir araya getirilerek aşağıdaki tablo çıkarılmıştır.

	Giriş Çıkış	Rastgele SS	Program Kullanımı	Görev	Key Mouse Logger	Ekran İzleme	Aktivite	URL	IdleTime	Gizli Modu	Personel durumu
Monitask	✓	✓	✓	✓	✓						
Desklog			✓			✓	✓	✓	✓		
Spector360		✓	✓					✓			
KickIdler	✓		✓		✓	✓		✓		✓	
Flexi Server	✓				✓			✓	✓		✓

Şekil 1.1 Piyasadaki Ürünlerin karşılaştırılması [1-5]

Özelliklerin açıklaması:

- Giriş/Çıkış: kullanıcının çalışma saatlerin içerisinde otomatik bir şekilde Giriş/Çıkış saatlerinin belirlenmesi.
- Rastgele SS(Rastgele Ekran görüntüsü): yazılım çalıştığı zamanlar içinde kontrol amaçlı rastgele bir şekilde ekran görüntüsünün alınıp kaydedilmesi.
- Program Kullanımı: her programın toplam çalışma süresinin tutulması.
- Görev: kullanıcının gün içerisinde yaptığı görevlerin girilip kaydedilmesi.
- Key/Mouse Logger: kullanıcının tüm bastığı tuşların dizi şeklinde tutulması.
- Ekran İzleme: belli bir aralıkta ekran görüntüsünün alınıp kaydedilmesi.
- Aktivite: mola verildi, çalışma başladı, program durduruldu gibi aktivitelerin kaydedilmesi.
- URL: kullanıcının girdiği tüm linklerin kaydedilmesi.
- IdleTime: kullanıcının çalışma saatlerinin alınan aksiyon sayışana göre pasif ve aktif zamana ayrılması.
- Gizli Modu: yazılımın kullanıcının haberi olmadan çalışması.
- Personel durumu: gerçek zamanda tüm kullanıcıların çalışma durumunun görülmesi.

1.2. İhtiyaç Analizi

Pandemi sürecinde 6 aydır çalıştığım GOSB Kule veri merkezinin farklı bölümlerini inceleyerek butik şirketlerde bu tip programları kullanarak çözülmesi gereken problemler aşağıdaki gibidir:

- (Yeni ortam.. Yeni çözüm) Pandemiden önce personel takibi yöneticiler tarafından ya izleyerek ya da çalışanlar ile konuşarak yapılıyordu. Yaptıkları işler fiziken yapıldığı için klasik takip yöntemler ile yapılması mümkündü. Fakat pandemi başladıktan sonra çalışanları yeni bir ortamda (uzaktan, dönüşümlü vb.) çalıştıkları için eski yöntemlerle takip etmek imkânsız hale geldi.
- (Doğru veri.. Doğru Karar) Çalışanların nasıl iş yaptıklarını anlamak ve analiz etmek büyük bir önem taşımaktadır. Fakat bu işi insanın yapması oldukça zordur. Çalışanların psikolojik durumu, yaptığı işi veya deneyimleri yaptıkları görevleri etkilemektedir. Bunları tespit etmek iş kalitesini arttırabilir.
- (Doğru kişi.. Doğru Görev) Aynı görevler zaman zaman farklı kişiler tarafından yapılmaktadır. Fakat hangi kişi görevini daha iyi bir şekilde tamamladığını anlamak her zaman kolay olmayabilir ve değerlendirme yapan kişinin belli bir tecrübeye sahip olması gerekmektedir.
- (Sıfır şüphe.. Sıfır sıkıntı) Uzaktan çalışıldığı için bazı çalışanlar yaptıkları iş veya mesaiye kaldıklarını kanıtlamakta sıkıntı çekmektedir.

1.3. Önerilen Çözüm

İhtiyaç analizindeki problemlerden yola çıkarak ve piyasadaki yazılımların özelliklerini göz önünde bulundurarak küçük şirketlere yönelik bir personel takip ve analiz sistemi geliştirilecektir. Bu sistem sayesinde çalışanların verilerini toplayarak her çalışanın gün sonunda neler ve ne zaman yaptığını tespit etmek mümkün olacak. Toplanacak veriler tamamen kullanıcıyı takip etmek değil çalışanlarda mevcut çalışma alışkanlıkları tespit edip çalışma trendlerini yakalamak amaçlıdır. Aynı anda bu verileri otomatik bir şekilde toplanıp kaydedileceği için yöneticiler tarafından analiz etmek kolay olacaktır. Program kullanıcıları rahatsız vermemek için kullanıcı

istediği zaman programı duraklatabilir ve kendi yaptığı işleri kanıtlamak için program sayesinde kolay olacaktır.

1.3.1. Yazılımın Özellikleri

Aşağıdaki verileri

1. Giriş/Çıkış Saatleri: program kullanıcı tarafında 24 saat çalıştığı için çalışanın çalışma saatleri tespit edilecek.
2. Program Kullanımı: çalışanın tüm kullandığı programlar zaman bilgisiyle birlikte kaydedilecek.
3. Domain Listesi: çalışanın tüm girdiği siteler zaman bilgisiyle birlikte kaydedilecek.
4. Aktif/Pasif Zaman: çalışanın gün içinde alınan aksiyon sayısına göre çalışma saatleri aktif/pasif olarak değerlendirilecek.
5. Aktiviteler: çalışanın çalıştığı saatler, mola verdiği dakikalar ve programı duraklattığı zamanlar kaydedilecek.
6. Klavye ve Mouse istatistikler: program bazlı çalışanların yaptıkları farklı girişler (Basılan klavye tuş sayısı, ortalama basma hızı, farklı tuş sayısı, backspace basma sayısı, fare tıklama sayısı, ortalama tıklama hızı) istatistik şeklinde kaydedilecek.
7. Görevler: çalışan istediği zaman yapmak istediği bir görev oluşturup otomatik bir şekilde sisteme kaydedilecek.
8. Kanıtlar: çalışan yaptığı işi kanıtlamak istediğinde ekran görüntüsünü yükleyerek sisteme dahil edilecek.

Toplayan bir yazılım geliştirilmiştir. Bu veriler sayesinde aşağıdaki amaçlar elde edilecektir:

1. (Yeni ortam.. Yeni çözüm): pandemiden dolayı yeni uzaktan çalışma sistemine geçen şirketler bu yazılım sayesinde otomatik ve çalışanları uğraştırmadan departman, kişi ve görev bazlı raporlar inceleyerek çalışanları ve iş akışını takip edebilecektir.
2. (Doğru veri.. Doğru Karar) toplanacak veriler çalışma verimliliğini aşağıdaki gibi arttıracaktır:

- Program/URL kullanımı: çalışanların hangi programları neden kullandığını analiz ederek gereksiz bir yazılımın maliyetinden kurtulabilir veya bir çalışanın yeni alınan yazılımın kullanmadığı takdirde bu problemi tespit edip nereden kaynaklandığı çözülebilir.
- Aktif/Pasif zaman: bu bilgiyi kullanarak çalışanlara saat ve gün bazlı performans değerlendirme yaparak her çalışanın en optimum çalışama zaman tespit edip ona göre iş vererek iş kalitesi arttırılabilir.
- Klavye ve Mouse istatistikleri: bu verileri kullanarak çalışanın yaptığı işin miktarı ölçmek kolay olacaktır. Ayrıca bu verileri kullanarak kullanıcıların psikolojik durumu tespit etmek mümkün olabilir.

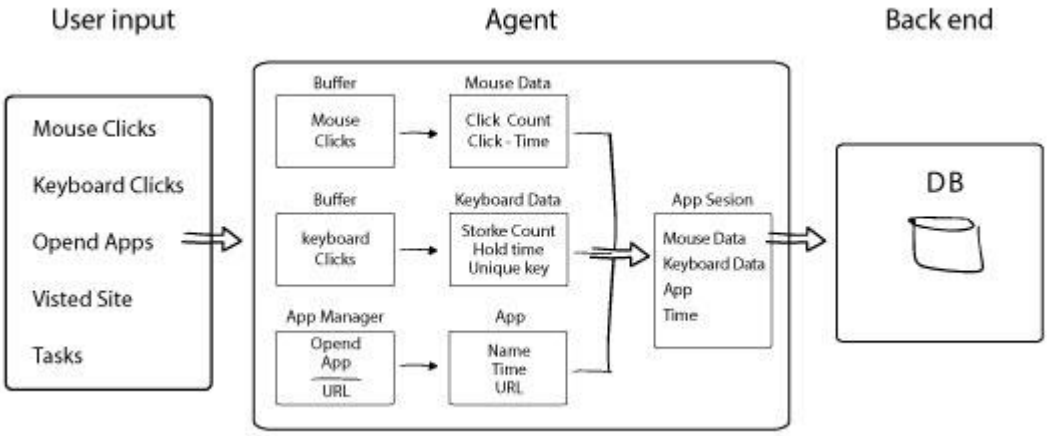
3. (Doğru kişi.. Doğru Görev) her kullanıcının oluşturduğu görevleri analiz ederek benzer görevleri yapan çalışanların verilerinin sistem üzerinden karşılaştırılması mümkün olacak. Bu veriler sayesinde her görevi tamamlamak için (best practice) tespit edilebilir ve doğru kişi doğru göreve yönlendirilebilir.
4. (Sıfır şüphe.. Sıfır sıkıntı): çalışan tarafından yüklenecek kanıtlar sayesinde yaptıkları işleri otomatik bir şekilde kanıtlanmış olacak ve fazla iş yaptıklarında o mesai saatlerini hesaplama mümkün olacak.

Geliştirilen yazılımın diğer programlara göre katkıları:

1. Çalışanın sürekli izleniyormuş gibi hissetmemesi için geliştirilmiş agent yazılımı istediği zaman duraklatabilir ve yazılım çalıştığı halde arka planda dikkatini dağıtmadan çalışacaktır.
2. Yazılım küçük ve yeni uzaktan çalışma sistemine geçen şirketlere yönelik kapsamlı bir şeklide geliştirilmiştir.
3. İş akışını iyileştirmek ve çalışan-yönetici arasında problemleri en aza indirmek hem çalışanı hem de iş vereni düşünerek geliştirilmiştir.
4. Keylogger özelliği benim geliştirdiğim sistem ile güvenlik açığı olmadan klavye ve Mouse verileri alınıp analiz edilecektir.

2. MATERYAL VE METOD

2.1.Çalışmanın Blok Şeması



Şekil 2.1. Blok Şeması

2.2. Çalışmanın Ana Bileşenleri

Yukarıda belirtilen çalışma aşağıdaki üç ana bileşenden oluşmaktadır:

1. HRP Monitor App: Client tarafında çalışacak agent programıdır. Bu program sayesinde yazılım özellikleri bölümünde bahsedilen verileri toplayıp düzenleyip ana sunucuya gönderecek. (1. Etap)
2. HRPM Web Service: ana sunucuda çalışan web servisidir. Bu web servisi sayesinde agent tarafından gelen verileri düzenlenip veri tabanına kaydedilecek. Aynı anda veri tabanındaki verileri raporlamak ve analiz etmekte yardımcı olacaktır. (1. Etap)
3. HRPMonitoring Web App: sunucudan çalışan web sitesidir. Bu site sayesinde hem çalışanları kendi yaptıkları işlemler hem de yöneticiler kendi çalışanlarının takibi ve analizini yapabilecektir. (2. Etap)

2.2.1. HRP Monitor App

Bir izleme ve takip programıdır. Bu program 24 saat boyunca hazır beklemektedir. Otomatik bir şekilde veya çalışan tarafından çalıştırıldığı halde kullanıcının verilerini toplamaktadır. Bu verileri benim oluşturduğum AppSession listesi yapısında

saklanmaktadır. Bu yapı kullanıcının kullandığı programlar, siteler ve klavye/mouse verilerini içermektedir. Ayrıca çalışan bir görev yapmak istediğinde programın ara yüzünden oluşturabilmektedir. Program hem online hem offline modunda çalışabildiği için bir loglama ve cache sistemine sahiptir. Aynı anda program bir sürü işlem yaptığı için bütün işler paralel (asynchronous) bir şekilde yapmaktadır. Ek olarak kullanıcının dikkatini dağıtmamak için basit ve araka planda (app tray içersinde) çalışabilen bir ara yüze sahiptir aynı anda çalışanı bilgilendirmek adına program çalışıp çalışmadığı net bir şekilde gözükmektedir. Programın belirtilen işleri gerçekleştiren 2 bileşeni var. Onlar:

2.2.1.1 HRPMCore

Programın çekirdek kütüphanesidir. Bu kütüphane windows32 API [6] ve Windows Hook'lerini [7] kullanarak istenen verileri almaktadır. Ayrıca program çalışma durumu ve veri manipülasyon işlemini yönetmektedir. Yapılan işlemler ve veriler birden fazla fonksiyon tarafından kullanıldığı için ve her class'ın sadece ve sadece bir tane instence'i olabilmesi için çoğu class'ta Singleton design pattern [8] kullanılmaktadır.

Kütüphanenin yaptığı ana işlemler:

1. Windows32 API'sindeki SetWindowsHookEx [9] fonksiyonunda Windows hook'ların (WH_KEYBOARD_LL, WH_MOUSE_LL [10]) kullanarak hem Mouse hem de klavye basılma olaylarına callback [11] fonksiyonlarıyla erişim sağlanmaktadır.
2. Program ve URL kullanımı elde etmek. Bu işi AppManager classı geliştirilerek gerçekleşmiştir. Her bir giriş aksiyonu (klavye ve mouse) alındığı zaman GetForegroundWindow [12] fonksiyonu kullanarak program değişip değişmediği kontrol ediliyor. Program değiştiği zaman bir önceki program için bir AppSession yapısında ilgili bilgiler saklanmakta ve yeni programın tipi ve bir tarayıcı ise açık olan link de alınmaktadır.

```
public class App
    {
        public string ExcutableName { get; set; }
        public string ProcessName { get; set; }
        public string HeaderText { get; set; }
        public string Content { get; set; }
        public AppType Type { get; set; }
    }
```

Şekil 2.2. App Class'ın Yapısı

3. Klavye kullanım verilerini elde etmek. Bu iş KeystrokesManager classı sayesinde yapılmaktadır. Her klavye tuş basılma tipi ve zamanı alınıp buffer listesinde saklanmaktadır. AppSession değiştiği zaman buffer'in içindeki veriler (Toplam basılma sayısı, toplam basılma süresi, basılan farklı tuş sayısı, backspace tuşu basılma sayısı) değerlendirilip kaydedilip ilgili session'a aktarılmaktadır.

```
public class KeyboardData
    {
        public int StrokesCount { get; set; }
        public int StrokeHoldTimes { get; set; }
        public int UniqueKeysCount { get; set; }
        public int BackspaceStrokesCount { get; set; }
    }
```

Şekil 2.3. KeyboardData Class'ın Yapısı

4. Mouse kullanım verilerini elde etmek. Bu iş MouseManager classı sayesinde yapılmaktadır. Her mouse tuş tıklama tipi ve zamanı alınıp buffer listesinde saklanmaktadır. AppSession değiştiği zaman buffer'in içindeki veriler (Toplam tıklama sayısı, toplam tıklama süresi) değerlendirilip kaydedilip ilgili session'a aktarılmaktadır.

```
public class MouseData
{
    public uint MouseClickCount { get; set; }
    public uint MouseClickTotalTime { get; set; }
}
```

Şekil 2.4. MouseData Class'ın Yapısı

5. Aktif/Pasif zamanları elde etmek. Bu iş TimeManager classı sayesinde yapılmaktadır. Her bir giriş aksiyonu (klavye ve mouse) alındığı zaman bir

önceki aksiyon zamana bakılıyor ve belli bir zaman aralığı aşmadığı zaman aktif zaman olarak aksi takdirde pasif zaman olarak kaydedilmektedir.

```
public class UsageTime
    {
        public double IdleMinutes { get; set; }
        public double ActiveMinutes { get; set; }
        public DateTime StartTime { get; set; }
        public DateTime EndTime { get; set; }
    }
```

Şekil 2.5. UsageTime Class'ın Yapısı

6. Çalışma durumunu kontrol etmek. StateController class'ları sayesinde tek bir yerde arka planda çalışan işlemler ve hook'lar yönetilmektedir. IStateController interface'i ve StateControllerManager class'ı sayesinde kodda bir değişiklik yapmadan istenen veri (App, klavye ve mouse) kullanılmaktadır.

2.2.1.2 HRPMonitor

Programın ara yüzüdür. Bu ara yüz üzerinden programı başlatma, duraklatma, çalışma durumunu öğrenme, görev oluşturma ve görev tamamlama işlemleri gerçekleştirilmektedir. Ara yüz yazılımı C# diliyle WPF platformunda MVVM [13] design pattern'i Calibro.Macro [14] kütüphanesi yardımıyla kullanarak geliştirilmiştir. Program tasarımı çalışmaya özgü olsun diye her şeyi pencere tasarımı dahil sıfırdan yapılmıştır. Program birden fazla dil seçeneğini desteklemektedir. Program çalışma saatlerine göre otomatik bir çalışma sistemine sahiptir. Programın arka planda çalışabilmesi için ve TaskBar'da ikonuun görünmemesi için Hardcodet [15] kütüphanesini kullanarak programın System Tray içine eklenmesi sağlanmıştır. Program çalışama durumu her değiştiğinde bildirim göndermektedir.

Ara yüzü üzerinde yapılan ana işlemler:

1. Kullanıcıya programın çalışma durumunu bildirmek.
2. Görevler oluşturup tamamlamak. Bir görev tamamlandıktan sonra APIHelper class'ı yardımıyla o görevin bilgileri API'ye gönderilmektedir.
3. Programı (HRPMCore) otmatik bir şekilde çalıştırıp durdurmak.
4. Programı (HRPMCore) kullanıcı isteğine göre çalıştırıp durdurmak.
5. Her 10 saniye oluşturulan logları API'ye göndermek.

6. Her 30Dk güncel AppSession listesini API'ye göndermek.

Cache Sistemi

Program ağ üzerinde verileri aktardığı için bağlantı ile ilgili problemler yaşanmaktadır. Bu problemler yüzünden veri kaybı oluşabilir. Bu kaybı önlemek için bir yedekleme sistemi geliştirilmiştir. FileManager class'ı sayesinde dosyalar sunucuya gönderme sürecinde bir kopma veya sunucu hatası olursa o dosyalar yerli diskte kaydedilip bağlantı çalışır durumuna geldiğinde tekrar sunucuya gönderilecektir.

2.2.2. HRPM Web Service

Sunucuda çalışan web servisidir. Bu web servisi sayesinde çalışmanın farklı bileşenlerinin arasında iletişim sağlanacaktır. Bu servis agent tarafından gelen verileri veya dosyaları işleyip hem diske hem de veri tabanına kaydedecektir. Aynı zamanda bütün veri tabanı işlemleri onun üzerinden yapılacaktır. Servis ASP.net platformunda WebAPI olarak geliştirilmiştir ve Windows Server sunucunda çalışmaktadır. Servisin belirtilen işleri gerçekleştiren 3 bileşeni var:

2.2.2.1 HRPMWebAPI

Bir ASP.net WebAPI [16] projesidir. Bu yazılım sayesinde API'nin linkleri belirlenip ilgili controller'da istekler işlenmektedir.

API üzerinden yapılan ana işlemler:

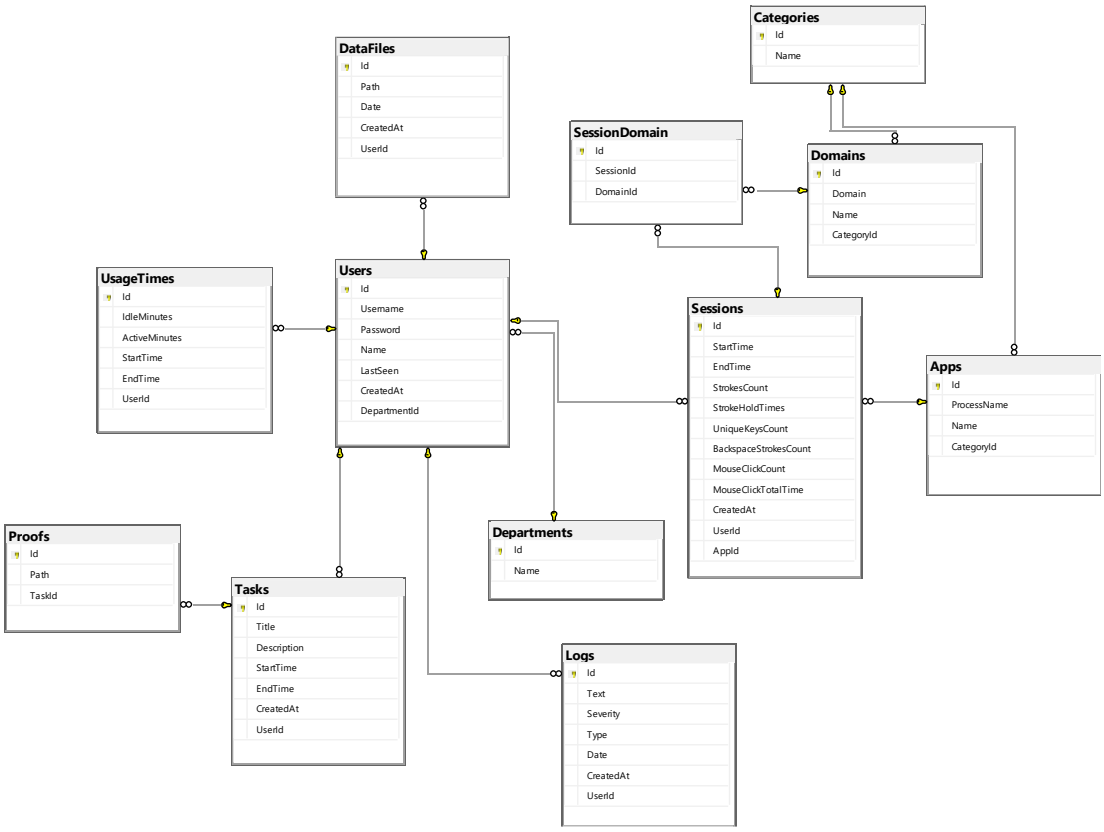
1. api/files/UploadFile linkinden gelen session verilerini veri tabanına ve dosyaları diske kaydetmek.
2. api/logs/PostLog linkinden gelen log verilerini veri tabanına kaydetmek.
3. api/tasks/PostTask linkinden gelen görev verilerini veri tabanına kaydetmek.
4. api/usagetimes/postusage linkinden gelen aktif/pasif zaman verilerini veri tabanına kaydetmek.
5. Doğrulama işlemleri.

2.2.2.2 HRPMBackendLibrary

Bu kütüphane sayesinde Dapper yazılımını kullanarak bütün veri tabanı sorguları yapılmaktadır. IDataConnection interface'ini kullanarak veri tabanı tipinden bağımsız bir yazılım geliştirilmiştir.

2.2.2.3 Veri Tabanı

Bütün çalışma yazılımları Windows ortamında çalıştığı için veri tabanı tipi MS-SQL olarak seçilmiştir. Aşağıda ER diyagramı bulunmaktadır.



Şekil 2.6. ER Diyagramı

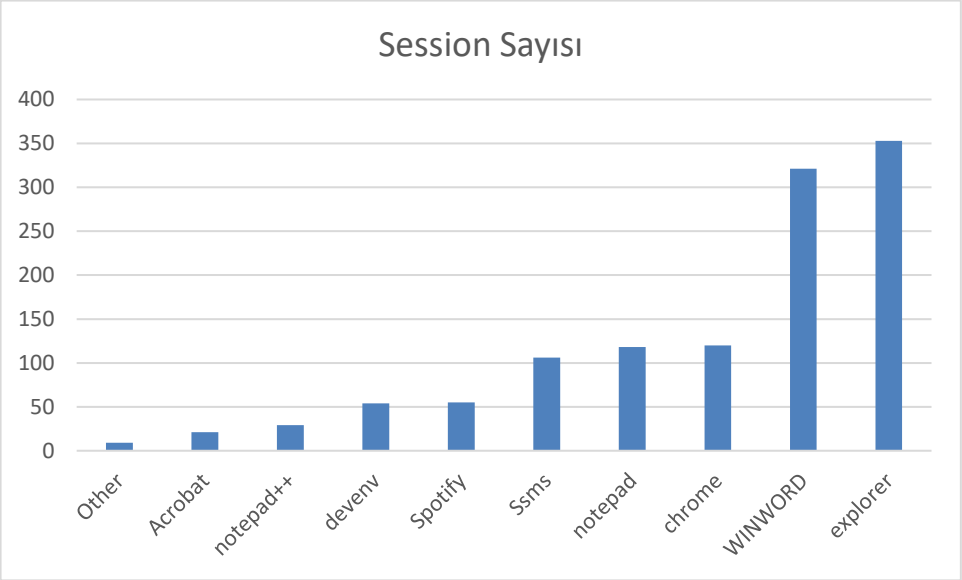
3. DENEYSEL ÇALIŞMALAR

3.1. Yapılan deneylerin değerlendirilmesi

Agent programı, API ve veri tabanı çalışır duruma geldikten sonra toplanacak verileri analiz etmek için bir gurup kullanıcılar bilgisayarlında program çalıştırılacaktır fakat çalışma zamanının dar olması ve gönüllü kullanıcılara programı kurdurtmanın zor olması için program tek kullanıcıda 24 saat boyunca çalıştırıldı ve aşağıdaki veriler elde edilmiştir.

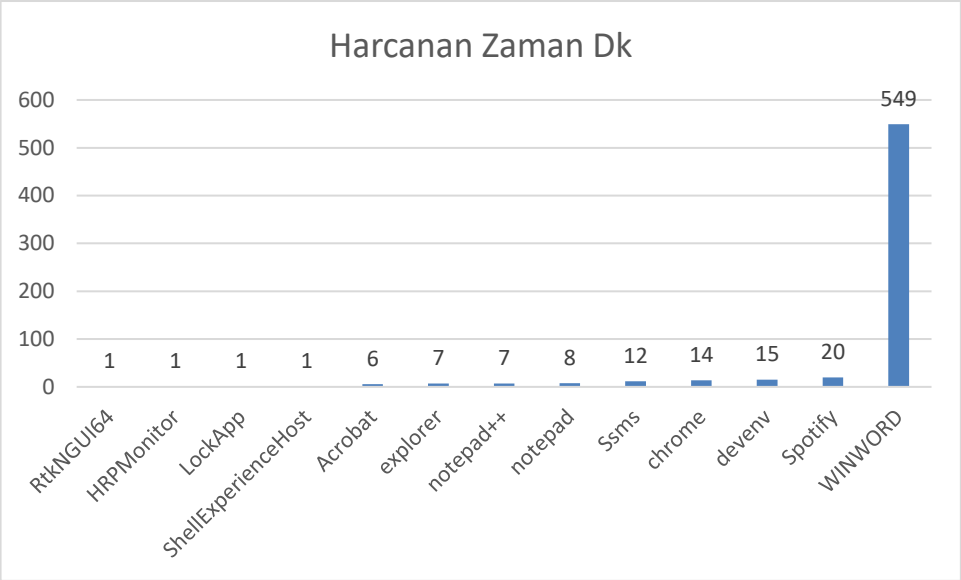
3.2. Session Verilerinin değerlendirilmesi

Session sayısı incelendiği zaman Explorer programı ve Word programı aynı oranda çalıştırıldığı ön görülmektedir.



Şekil 3.1. Program Bazlı Session Sayısı Grafiği

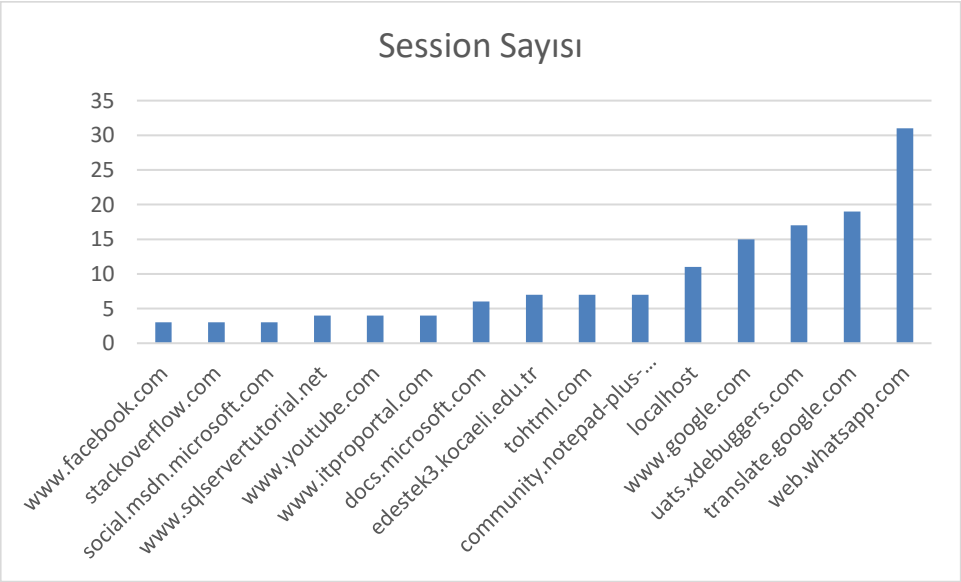
Fakat harcanan zaman verileri incelendiğinde Word programı çalıştırma süresi daha çok olduğu görülmektedir. Bundan anlaşılan şu ki: kullanıcı Explorer programını çok fazla kullanıyor lakin fazla zaman harcamıyor.



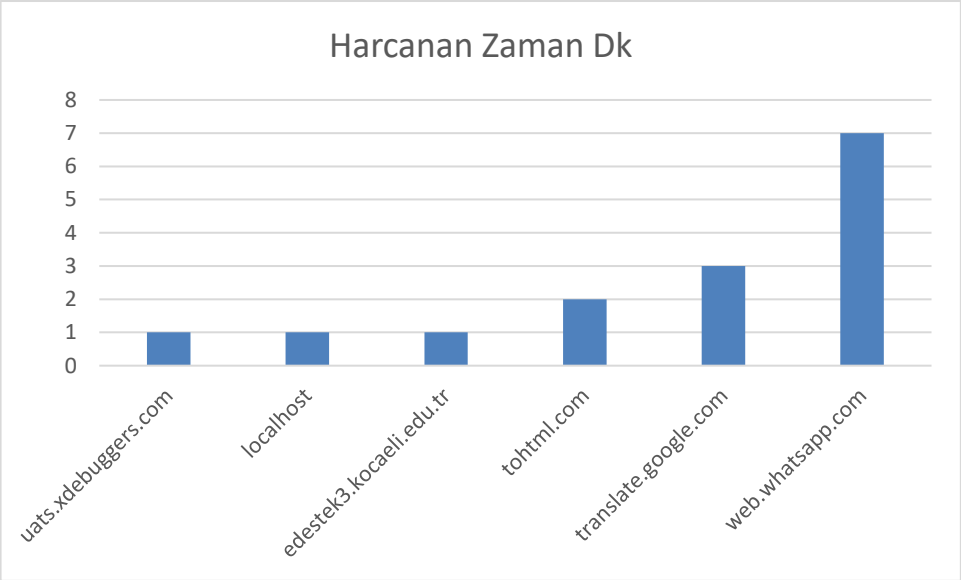
Şekil 3.2. Program Bazlı Session'lerde Harcanan Zamanlar Grafiği

3.3. URL Verilerinin değerlendirilmesi

Aşağıdaki tablolar incelendiği zaman farklı bir davranış görülmektedir. Bazı sitelerin session sayısı ve session süreleri aynı oranda gösterilmektedir. Bundan anlaşılan şu ki: kullanıcı Whatsapp sitesini çok kullanıyor ve aynı anda fazla zaman harcıyor.



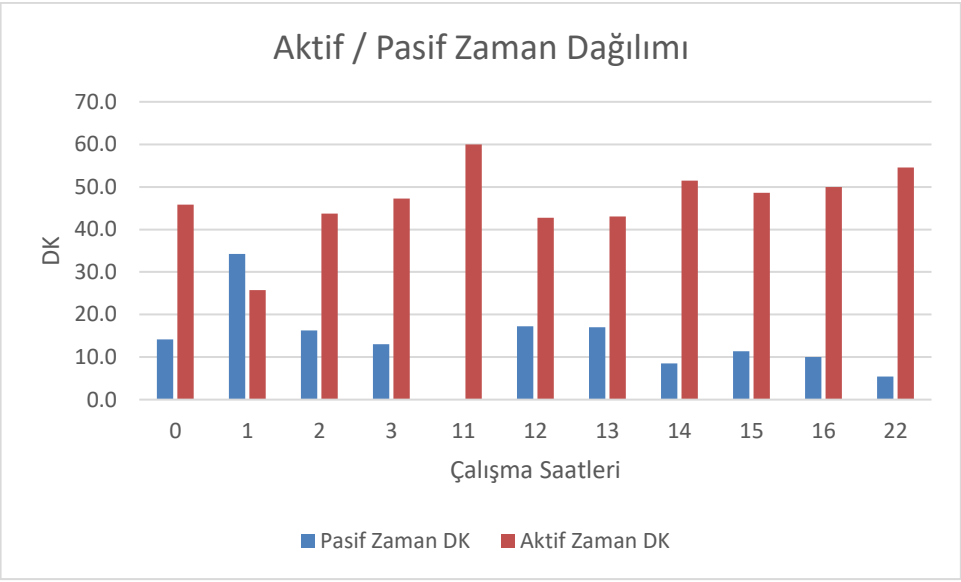
Şekil 3.3. URL Bazlı Session Sayısı Grafiği



Şekil 3.4. URL Bazlı Session'lerde Harcanan Zamanlar Grafiği

3.4.Aktif / Pasif Zamanları Verilerinin değerlendirilmesi

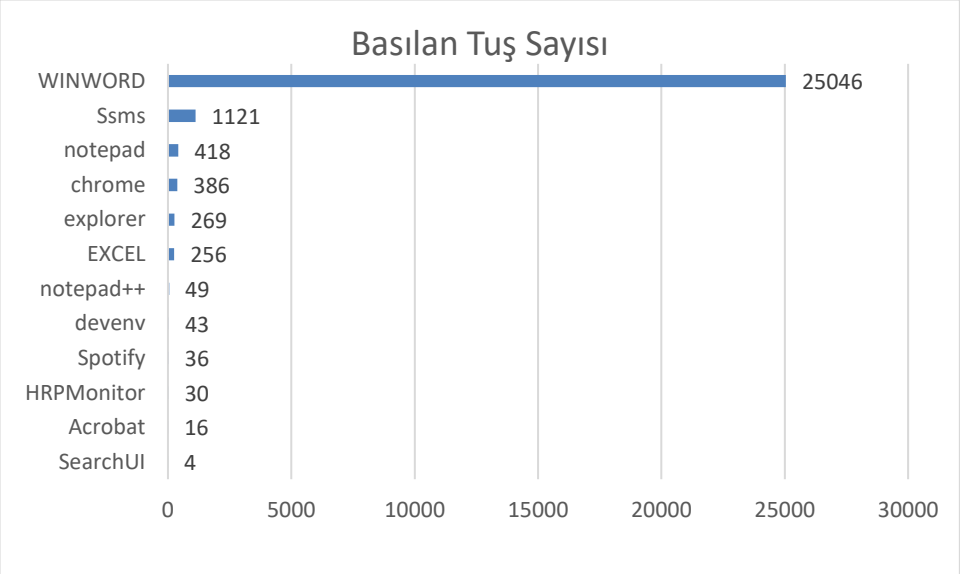
Aktif / Pasif zaman dağılımına bakıldığı zaman kullanıcı sabah saatlerinde daha verimli bir şekilde çalıştığı görülmektedir.



Şekil 3.5. 24 Saat içinde Aktif / Pasif Zaman Dağılım Grafiği

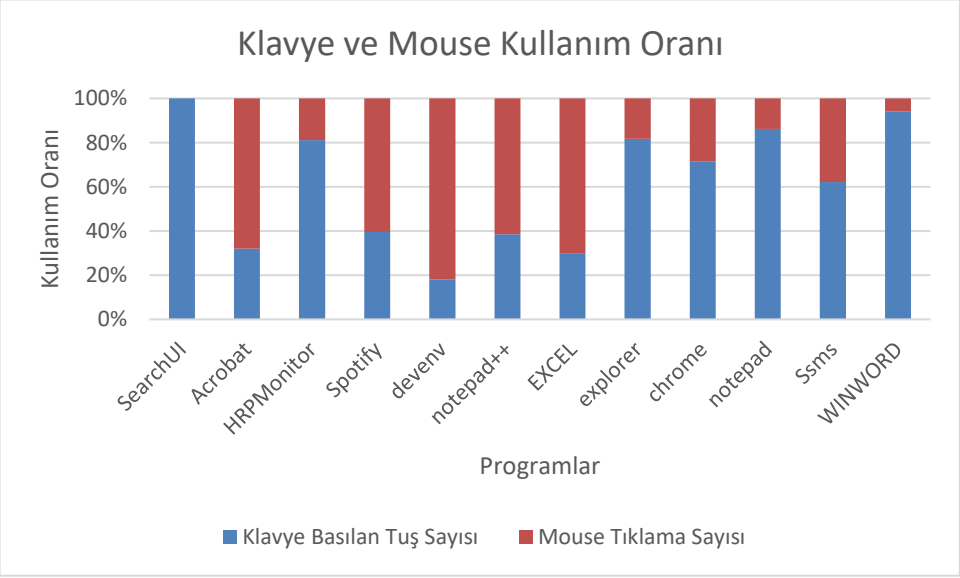
3.5. Klavye ve Mouse Kullanım Verilerinin değerlendirilmesi

Basılan tuş sayısına bakıldığı zaman kullanıcının her programda ne kadar çalıştığını anlamak mümkündür. Ve bu verileri diğer tablo verileri ile karşılaştırıldığı zaman kullanıcının Word programını çok uzun sürede uzun bir yazı girdiği anlaşılmaktadır.



Şekil 3.6. Program Bazlı Klavye Basılan Tuş Sayısı Grafiği

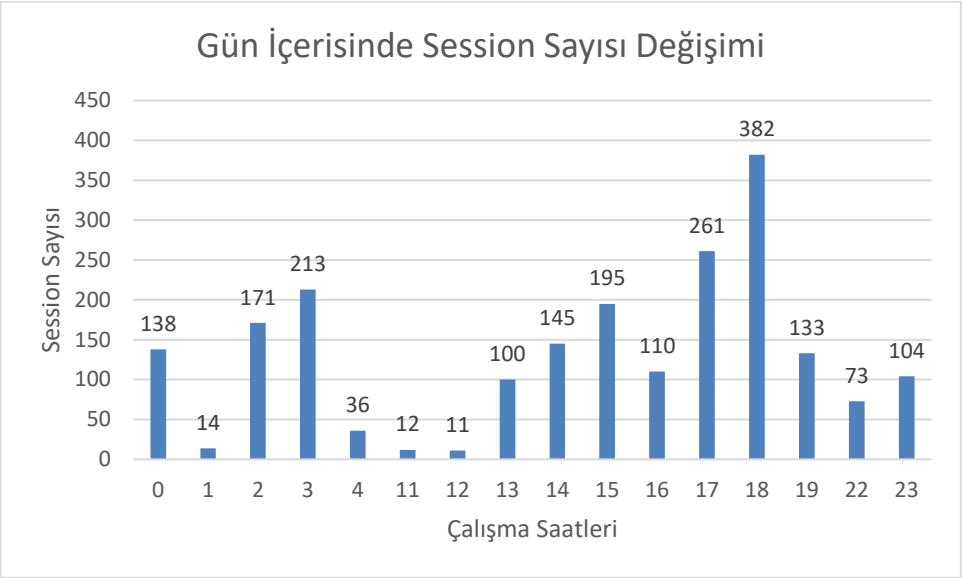
Klavye ve Mouse program bazlı kullanım oranları takip edilerek bir donanım problemi veya program yanlış kullanımı ortaya çıkabilmektedir.



Şekil 3.7. Program Bazlı Klavye ve Mouse Kullanım Oranı Grafiği

3.6. Birden Fazla Veri Kullanarak Çıkan Verilerin değerlendirilmesi

Session sayısını Aktif / Pasif zamanları ile karşılaştırılarak takip edildiği zaman kullanıcı ne kadar konsantre olduğu görülmektedir. Aşağıdaki verilerden kullanıcının saat 11 ve 22 iyi bir performans yaptığı farkedilmektedir.



Şekil 3.8. 24 Saat İçinde Session Sayısı Değişim Grafiği

4. SONUÇLAR

Deneysel çalışmalar bölümünde bahsedildiği gibi fazla veri toplanamamıştır. Bu sebepten dolayı bazı program fonksiyonları kullanılamamıştır. Fakat bu fonksiyonlar faydasız anlamına gelmez. Çıkan sonuçları ve toplanmış veriler programın doğru bir şekilde hata olmadan çalıştığını göstermektedir. Bu çalışmanın en iyi sonuçları verebilmek için gerçek bir firmada denenmesi gerekmektedir. Çalışama zamanı dar olduğu için ve daha kapsamlı bir agent yazılımını gerçekleştirmek için çalışma iki etaba bölünmüştür. 1. Etapta agent yazılımı, api ve database kurumu gerçekleştirilmiştir. 2. Etapta toplanan verileri analize eden, raporlayan ve karar alma yardımcı olacak Web App yazılımı geliştirilecektir.

KAYNAKLAR

[1] Monitask - Employee Monitoring, https://www.monitask.com/en

[2] Desklog - Automated Employee Monitoring Software, https://desklog.io/

[3] SPECTOR 360, https://www.monitortools.com/software/product/spector-360/

[4] KickIdler - Employee Monitoring Software, https://www.kickidler.com/

[5] FlexiServer - Productivity & Attendance Software, https://www.nchsoftware.com/flexi/index.html

[6] Windows API index, https://docs.microsoft.com/en-us/windows/win32/apiindex/windows-api-list

[7] Windows Hooks, https://docs.microsoft.com/en-us/windows/win32/winmsg/hooks#:~:text=A%20hook%20is%20a%20point,reach%20the%20target%20window%20procedure

[8] Singleton Design Pattern, https://www.geeksforgeeks.org/singleton-design-pattern/

[9] SetWindowsHookExA function (winuser.h), https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-setwindowshookexa

[10] Hooks Overview, https://docs.microsoft.com/en-us/windows/win32/winmsg/about-hooks

[11] LowLevelKeyboardProc callback function, https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/ms644985(v=vs.85)

[12] GetForegroundWindow function (winuser.h), https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getforegroundwindow

[13] Model-View-ViewModel (MVVM) Explained, Jeremy Likness, https://www.wintellect.com/model-view-viewmodel-mvvm-explained/

[14] Caliburn.Micro, https://caliburnmicro.com/documentation/

[15] WPF NotifyIcon, http://www.hardcodet.net/wpf-notifyicon

[16] ASP.NET Web APIs, https://dotnet.microsoft.com/apps/aspnet/apis

```
namespace HRPMBackendLibrary.DataAccess
{
    public class FileManager
    {
        private static readonly FileManager _instance = new FileManager();
        private readonly string _rootFolder = @"C:\Users\mhdb9\Desktop\API\";
        private readonly string _rootKeystrokesFolder = @"Data\";
        private FileManager()
        {
            Directory.CreateDirectory(_rootFolder);
        }
        public static FileManager GetFileManager()
        {
            return _instance;
        }
        private void WriteBytesToDisk(byte[] bytes, string path)
        {
            try
            {
                File.WriteAllBytes(path, bytes);
            }
            catch (Exception)
            {
                throw;
            }
        }
        public string SaveSessionsFile(BinaryFile file)
        {
            string path = $@"{_rootFolder}{_rootKeystrokesFolder}{file.User.UserName}";
            Directory.CreateDirectory(path);
            try
            {
                path += $@"\{DateTime.Now:yyyy-MM-dd}-{file.Name}";
                WriteBytesToDisk(file.Bytes, path);
                return path;
            }
            catch (Exception)
            {
                throw;
            }
        }
    }
}
namespace HRPMBackendLibrary.DataAccess
{
    public class SqlConnector : IDataConnection
    {
        public static string databaseName = "HRPM";
        public void File_Insert(DataFileModel model)
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@Path", model.Path);
                p.Add("@UserId", model.User.Id);
                p.Add("@Date", model.Date);
                p.Add("@id", 0, dbType: DbType.Int32, direction: ParameterDirection.Output);
                connection.Execute("dbo.spFiles_Insert", p, commandType: CommandType.StoredProcedure);
                model.Id = p.Get<int>("@id");
            }
        }
        public void Key_Insert(KeysList key)
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@Id", (int)key);
                p.Add("@Code", key.ToString());
                p.Add("@Name", key.GetDescription());
                connection.Execute("dbo.spKeys_Insert", p, commandType: CommandType.StoredProcedure);
            }
        }
        public void Log_Insert(LogModel model)
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@Text", model.Text);
                p.Add("@UserId", model.User.Id);
```

```
                p.Add("@Type", model.Type.GetDescription());
                p.Add("@Date", model.Date);
                p.Add("@Severity", model.Severity.GetDescription());
                p.Add("@id", 0, dbType: DbType.Int32, direction: ParameterDirection.Output);
                connection.Execute("dbo.spLogs_Insert", p, commandType: CommandType.StoredProcedure);
                model.Id = p.Get<int>("@id");
            }
        }
        public bool User_Exists(UserModel model)
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                List<UserModel> users = new List<UserModel>();
                var p = new DynamicParameters();
                p.Add("@UserName", model.Username);
                users = connection.Query<UserModel>("dbo.spUsers_GetUserByUserName", p, commandType:
CommandType.StoredProcedure).ToList();
                if (users.Any())
                {
                    model.Id = users.FirstOrDefault().Id;
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }
        public void User_Insert(UserModel model)
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@UserName", model.Username);
                p.Add("@id", 0, dbType: DbType.Int32, direction: ParameterDirection.Output);
                connection.Execute("dbo.spUsers_Insert", p, commandType: CommandType.StoredProcedure);
                model.Id = p.Get<int>("@id");
            }
        }
        public List<UserModel> User_SelectAll()
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                List<UserModel> users = new List<UserModel>();
                users = connection.Query<UserModel>("dbo.spUsers_SelectAll").ToList();
                return users;
            }
        }
        public List<DataFileModel> File_SelectAll()
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                List<DataFileModel> files = new List<DataFileModel>();
                files = connection.Query<DataFileModel, UserModel, DataFileModel>("dbo.spFiles_SelectAll",
                   (file, user)
                   =>
                   {
                       file.User = user;
                       return file;
                   }).ToList();
                return files;
            }
        }
        //GetUserUploadFiles
        public List<DataFileModel> GetFiles_BySearchValue(int searchValue)
        {
            List<DataFileModel> output;
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@UserId", searchValue);
                output = connection.Query<DataFileModel, UserModel, DataFileModel>("dbo.spFiles_GetByUserName",
                    (file, user) =>
                    {
                        file.User = user;
                        return file;
                    }, p, commandType: CommandType.StoredProcedure).ToList();
            }
            return output;
```

```
        }
        public List<DataFileModel> GetFiles_BySearchDate(string searchValue)
        {
            List<DataFileModel> output;
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                DateTime dateTime = DateTime.Parse(searchValue);
                p.Add("@Date", searchValue);
                output = connection.Query<DataFileModel, UserModel, DataFileModel>("dbo.spFiles_GetByDate",
                     (file, user) =>
                     {
                         file.User = user;
                         return file;
                     }, p, commandType: CommandType.StoredProcedure).ToList();
            }
            return output;
        }
        public List<LogModel> Log_SelectAll()
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                List<LogModel> logs = new List<LogModel>();
                logs = connection.Query<LogModel, UserModel, LogModel>("dbo.spLogs_SelectAll",
                   (log, user)
                   =>
                   {
                       log.User = user;
                       return log;
                   }).ToList();
                return logs;
            }
        }
        public List<LogModel> GetLogs_BySearchValue(int searchValue)
        {
            List<LogModel> output;
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@UserId", searchValue);
                output = connection.Query<LogModel, UserModel, LogModel>("dbo.spLogs_GetByUserName",
                    (log, user) =>
                    {
                        log.User = user;
                        return log;
                    }, p, commandType: CommandType.StoredProcedure).ToList();
            }
            return output;
        }
        public List<LogModel> GetLogs_BySearchDate(string searchValue)
        {
            List<LogModel> output;
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                DateTime dateTime = DateTime.Parse(searchValue);
                p.Add("@Date", searchValue);
                output = connection.Query<LogModel, UserModel, LogModel>("dbo.spLogs_GetByDate",
                     (log, user) =>
                     {
                         log.User = user;
                         return log;
                     }, p, commandType: CommandType.StoredProcedure).ToList();
            }
            return output;
        }
        public List<UserModel> GetUsers_BySearchLogType(string searchValue)
        {
            List<UserModel> output;
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@Type", searchValue);
                output = connection.Query<UserModel>("dbo.spUsers_GetByLogType", p, commandType:
CommandType.StoredProcedure).ToList();
            }
            return output;
        }
        public List<LogModel> GetLogs_BySearchUser(int searchValue)
```

```
{
    List<LogModel> output;
    using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
    {
        var p = new DynamicParameters();
        p.Add("@UserId", searchValue);
        output = connection.Query<LogModel, UserModel, LogModel>("dbo.spLogs_GetByUser",
             (log, user) =>
             {
                 log.User = user;
                 return log;
             }, p, commandType: CommandType.StoredProcedure).ToList();
    }
    return output;
}
public List<LogModel> GetLogs_AllLogs()
{
    List<LogModel> output;
    using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
    {
        output = connection.Query<LogModel, UserModel, LogModel>("dbo.spLogs_SelectAll",
             (log, user) =>
             {
                 log.User = user;
                 return log;
             }, commandType: CommandType.StoredProcedure).ToList();
    }
    return output;
}
public List<DataFileModel> Get_AllFiles()
{
    List<DataFileModel> output;
    using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
    {
        output = connection.Query<DataFileModel, UserModel, DataFileModel>("dbo.spFiles_SelectAll",
             (file, user) =>
             {
                 file.User = user;
                 return file;
             }, commandType: CommandType.StoredProcedure).ToList();
    }
    return output;
}
public List<LogModel> GetLogs_ByLogTypeAndUserId(int searchValue, string type)
{
    List<LogModel> output;
    using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
    {
        var p = new DynamicParameters();
        p.Add("@UserId", searchValue);
        p.Add("@Type", type);
        output = connection.Query<LogModel, UserModel, LogModel>("dbo.spLogs_GetByLogTypeAndUser",
            (log, user) =>
            {
                log.User = user;
                return log;
            }, p, commandType: CommandType.StoredProcedure).ToList();
    }
    return output;
}
public List<LogModel> GetLogs_ByLogType(string type)
{
    List<LogModel> output;
    using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
    {
        var p = new DynamicParameters();
        p.Add("@Type", type);
        output = connection.Query<LogModel, UserModel, LogModel>("dbo.spLogs_GetByLogType",
            (log, user) =>
            {
                log.User = user;
                return log;
            }, p, commandType: CommandType.StoredProcedure).ToList();
    }
    return output;
}
public DataFileModel GetFilesById(int id)
{
    DataFileModel output;
```

```
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@FileId", id);
                output = connection.Query<DataFileModel>("dbo.spFiles_GetById", p, commandType:
CommandType.StoredProcedure).ToList().FirstOrDefault();
            }
            return output;
        }
        public List<DepartmentModel> Department_SelectAll()
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                List<DepartmentModel> deps = new List<DepartmentModel>();
                deps = connection.Query<DepartmentModel>("dbo.spDepartments_SelectAll").ToList();
                return deps;
            }
        }
        public List<UserModel> User_GetByDepartment(int id)
        {
            List<UserModel> output;
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@DepartmentId", id);
                output = connection.Query<UserModel>("dbo.spUsers_GetByDepartmentId",
                       p, commandType: CommandType.StoredProcedure).ToList();
            }
            return output;
        }
        public string User_GetPassword(int userId)
        {
            UserModel output;
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@UserId", userId);
                output = connection.Query<UserModel>("spUsers_GetPassword",
                       p, commandType: CommandType.StoredProcedure).FirstOrDefault();
            }
            return output.Password;
        }
        public void App_Insert(AppModel model)
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@ProcessName", model.ProcessName);
                p.Add("@id", 0, dbType: DbType.Int32, direction: ParameterDirection.Output);
                connection.Execute("dbo.spApps_Insert", p, commandType: CommandType.StoredProcedure);
                model.Id = p.Get<int>("@id");
            }
        }
        public void Domain_Insert(DomainModel model)
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@Domain", model.Domain);
                p.Add("@id", 0, dbType: DbType.Int32, direction: ParameterDirection.Output);
                connection.Execute("dbo.spDomains_Insert", p, commandType: CommandType.StoredProcedure);
                model.Id = p.Get<int>("@id");
            }
        }
        public void SessionDomain_Insert(int sessionId, int domainId)
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@SessionId", sessionId);
                p.Add("@DomainId", domainId);
                connection.Execute("spSessionDomain_Insert", p, commandType: CommandType.StoredProcedure);
            }
        }
        public void Session_Insert(SessionModel model)
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
```

```
                p.Add("@StartTime", model.StartTime);
                p.Add("@EndTime", model.EndTime);
                p.Add("@StrokesCount", model.StrokesCount);
                p.Add("@StrokeHoldTimes", model.StrokeHoldTimes);
                p.Add("@UniqueKeysCount", model.UniqueKeysCount);
                p.Add("@BackspaceStrokesCount", model.BackspaceStrokesCount);
                p.Add("@MouseClickCount", model.MouseClickCount);
                p.Add("@MouseClickTotalTime", model.MouseClickTotalTime);
                p.Add("@UserId", model.UserId);
                p.Add("@AppId", model.AppId);
                p.Add("@id", 0, dbType: DbType.Int32, direction: ParameterDirection.Output);
                connection.Execute("dbo.spSessions_Insert", p, commandType: CommandType.StoredProcedure);
                model.Id = p.Get<int>("@id");
            }
        }
        public void Task_Insert(TaskModel model)
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@Title", model.Title);
                p.Add("@Description", model.Description);
                p.Add("@StartTime", model.StartTime);
                p.Add("@EndTime", model.EndTime);
                p.Add("@UserId", model.UserId);
                p.Add("@id", 0, dbType: DbType.Int32, direction: ParameterDirection.Output);
                connection.Execute("spTasks_Insert", p, commandType: CommandType.StoredProcedure);
                model.Id = p.Get<int>("@id");
            }
        }
        public void UsageTime_Insert(UsageTimeModel model)
        {
            using (IDbConnection connection = new SqlConnection(GlobalConfig.CnnString(databaseName)))
            {
                var p = new DynamicParameters();
                p.Add("@ActiveMinutes", model.ActiveMinutes);
                p.Add("@IdleMinutes", model.IdleMinutes);
                p.Add("@StartTime", model.StartTime);
                p.Add("@EndTime", model.EndTime);
                p.Add("@UserId", model.UserId);
                p.Add("@id", 0, dbType: DbType.Int32, direction: ParameterDirection.Output);
                connection.Execute("spUsageTimes_Insert", p, commandType: CommandType.StoredProcedure);
                model.Id = p.Get<int>("@id");
            }
        }
    }
}
namespace HRPMBackendLibrary.Helpers
{
    public class DbHelper
    {
        public static void CheckForUser(UserModel model)
        {
            if (!GlobalConfig.Connection.User_Exists(model))
            {
                GlobalConfig.Connection.User_Insert(model);
            }
        }
        public static byte[] Serialize<T>(T data)
        {
            var binFormatter = new BinaryFormatter();
            var mStream = new MemoryStream();
            binFormatter.Serialize(mStream, data);
            return mStream.ToArray();
        }
        public static T Deserialize<T>(byte[] data) where T : class
        {
            var mStream = new MemoryStream();
            var binFormatter = new BinaryFormatter();
            // Where 'objectBytes' is your byte array.
            mStream.Write(data, 0, data.Length);
            mStream.Position = 0;
            return binFormatter.Deserialize(mStream) as T;
        }
    }
}
namespace HRPMBackendLibrary.Helpers
{
    public static class EnumHelper
```

```
    {
        public static List<string> GetValues<T>()
        {
            List<string> values = new List<string>();
            foreach (var itemType in Enum.GetValues(typeof(T)))
            {
                //For each value of this enumeration, add a new EnumValue instance
                values.Add(Enum.GetName(typeof(T), itemType));
            }
            return values;
        }
    }
}
namespace HRPMCore.Managers
{
    public class AppManager
    {
        private static readonly AppManager _instance = new AppManager();
        private bool isInitilized = false;
        public string filePath;
        private List<AppSession> sessions;
        private bool isBusy = false;
        public bool IsBusy {
            get
            {
                return isBusy;
            }
        }
        public List<AppSession> GetAllSessions()
        {
            if (isInitilized)
            {
                if (sessions == null)
                {
                    try
                    {
                        sessions = BinaryConnector.StaticLoad<List<AppSession>>(filePath);
                        return sessions;
                    }
                    catch (Exception)
                    {
                        sessions = new List<AppSession>();
                        return sessions;
                    }
                }
                else
                {
                    return sessions;
                }
            }
            else
            {
                throw new Exception("Data cache file path was not provided. Call AppManager.Initialize(string
path) to pass the path");
            }
        }
        public void ClearSessions()
        {
            sessions = null;
        }
        public void Initialize(string path)
        {
            isInitilized = true;
            filePath = path;
        }
        private AppManager()
        {
        }
        public static AppManager GetAppManager()
        {
            return _instance;
        }
        public AppSession GetLastSession()
        {
            if(GetAllSessions().Count > 0)
            {
                return GetAllSessions().Last();
            }
            else
```

```
    {
        return null;
    }
}
private void CreateSession(AppSession session)
{
    if(GetLastSession() != null)
    {
        if(GetLastSession().KeyboardData == null || GetLastSession().MouseData == null)
        {
            GetAllSessions().RemoveAt(GetAllSessions().Count-1);
        }
    }
    GetAllSessions().Add(session);
    SaveSessionsToDisk();
}
private void SaveSessionsToDisk()
{
    BinaryConnector.StaticSave(GetAllSessions(), filePath);
}
private WindowInfo GetWindowInfo()
{
    WindowInfo win = new WindowInfo();
    win.WindowHandler = NativeMethods.GetForegroundWindow();
    StringBuilder title = new StringBuilder(256);
    NativeMethods.GetWindowText(win.WindowHandler, title, title.Capacity);
    win.Title = title.ToString();
    return win;
}
public async Task<bool> /*bool*/ CheckIfSessionChanged()
{
    isBusy = true;
    bool isChanged;
    WindowInfo winInfo = GetWindowInfo();
    if (GetAllSessions().Count > 0)
    {
        AppSession lastSession = GetLastSession();
        if (winInfo.Title == lastSession.App.HeaderText || winInfo.Title == "")
        {
            isChanged = false;
        }
        else
        {
            AppSession newSession = await Task.Run(() => GetNewSessionInfo(winInfo));
            if (newSession == null)
            {
                isChanged = false;
            }
            else
            {
                SaveSessionData(lastSession);
                if (lastSession.KeyboardData == null || lastSession.MouseData == null)
                {
                    isChanged = false;
                }
                else
                {
                    //Console.WriteLine(newSession.App.ProcessName);
                    CreateSession(newSession);
                    isChanged = true;
                }
            }
        }
    }
    else
    {
        AppSession newSession = GetNewSessionInfo(winInfo);
        if (newSession == null)
        {
            isChanged = false;
        }
        else
        {
            CreateSession(newSession);
            isChanged = false;
        }
    }
    isBusy = false;
    return isChanged;
```

```
        }
        public void SaveSessionData(AppSession session)
        {
            if(session != null)
            {
                if (session.KeyboardData == null && session.MouseData == null)
                {
                    session.EndTime = DateTime.Now;
                    session.KeyboardData = KeystrokesManager.GetKeyStrokesManager().GetKeyboardData();
                    session.MouseData = MouseManager.GetMouseManager().GetMouseData();
                    KeystrokesManager.GetKeyStrokesManager().SessionChanged();
                    MouseManager.GetMouseManager().SessionChanged();
                }
            }
        }
        private AppSession GetNewSessionInfo(WindowInfo winInfo)
        {
            AppSession newSession = new AppSession();
            newSession.StartTime = DateTime.Now;
            newSession.App.HeaderText = winInfo.Title;
            Process foregroundProcess =
Process.GetProcessById(NativeMethods.GetWindowProcessId(winInfo.WindowHandler));
            if (foregroundProcess.ProcessName == "ApplicationFrameHost")
            {
                try
                {
                    if (foregroundProcess == null)
                    {
                        return null;
                    }
                    foregroundProcess = GetRealProcess(foregroundProcess);
                    newSession.App.ProcessName = foregroundProcess.ProcessName;
                }
                catch (Exception)
                {
                    return null;
                }
            }
            else
            {
                newSession.App.ProcessName = foregroundProcess.ProcessName;
                if (foregroundProcess.ProcessName == "chrome")
                {
                    newSession.App.Content =  GetUrl(winInfo.WindowHandler);
                    newSession.App.Type = AppType.Browser;
                }
            }
            try
            {
                newSession.App.ExcutableName = foregroundProcess.MainModule.FileName.ToString();
            }
            catch (Exception)
            {
                newSession.App.ExcutableName = "Access Denied";
            }
            return newSession;
        }
        private /*async Task<string>*/string GetUrl(IntPtr hWindow)
        {
            AutomationElement elm = AutomationElement.FromHandle(hWindow);
            AutomationElement elmUrlBar;
            try
            {
                //var t = elm.FindAll(TreeScope.Children, new
PropertyCondition(AutomationElement.ControlTypeProperty, ControlType.Pane));
                var pane1 = elm.FindAll(TreeScope.Children, new
PropertyCondition(AutomationElement.ControlTypeProperty, ControlType.Pane))[1];
                var pane2 = pane1.FindAll(TreeScope.Children, new
PropertyCondition(AutomationElement.ControlTypeProperty, ControlType.Pane))[1];
                var pane3 = pane2.FindFirst(TreeScope.Children, new
PropertyCondition(AutomationElement.ControlTypeProperty, ControlType.Pane));
                var pane4 = pane3.FindFirst(TreeScope.Children, new
PropertyCondition(AutomationElement.ControlTypeProperty, ControlType.Pane));
                var cus = pane4.FindFirst(TreeScope.Children, new
PropertyCondition(AutomationElement.ControlTypeProperty, ControlType.Custom));
                elmUrlBar = cus.FindFirst(TreeScope.Children, new
PropertyCondition(AutomationElement.ControlTypeProperty, ControlType.Edit));
            }
            catch
```

```
            {
                return "";
            }
            // make sure it's valid
            if (elmUrlBar == null)
            {
                // it's not..
                return "";
            }
            // elmUrlBar is now the URL bar element. we have to make sure that it's out of keyboard focus if we
want to get a valid URL
            if ((bool)elmUrlBar.GetCurrentPropertyValue(AutomationElement.HasKeyboardFocusProperty))
            {
                return "";
            }
            // there might not be a valid pattern to use, so we have to make sure we have one
            AutomationPattern[] patterns = elmUrlBar.GetSupportedPatterns();
            if (patterns.Length == 1)
            {
                string ret = "";
                try
                {
                    ret = ((ValuePattern)elmUrlBar.GetCurrentPattern(patterns[0])).Current.Value;
                    var uri = new Uri(ret);
                    ret = uri.Host;
                    return ret;
                }
                catch { }
                if (ret != "")
                {
                    // must match a domain name (and possibly "https://" in front)
                    if (Regex.IsMatch(ret, @"^(https:\/\/)?[a-zA-Z0-9\-\.]+(\.[a-zA-Z]{2,4}).*$"))
                    {
                        // prepend http:// to the url, because Chrome hides it if it's not SSL
                        if (!ret.StartsWith("http"))
                        {
                            ret = "http://" + ret;
                        }
                        //Console.WriteLine("Open Chrome URL found: '" + ret + "'");
                        return ret;
                    }
                }
                return "";
            }
            return "";
        }
        private Process _realProcess;
        private Process GetRealProcess(Process foregroundProcess)
        {
            NativeMethods.EnumChildWindows(foregroundProcess.MainWindowHandle, ChildWindowCallback,
IntPtr.Zero);
            return _realProcess;
        }
        private bool ChildWindowCallback(IntPtr hwnd, IntPtr lparam)
        {
            var process = Process.GetProcessById(NativeMethods.GetWindowProcessId(hwnd));
            if (process.ProcessName != "ApplicationFrameHost")
            {
                _realProcess = process;
            }
            return true;
        }
    }
}
namespace HRPMCore.Managers
{
    public class KeystrokesManager
    {
        private static readonly KeystrokesManager _instance = new KeystrokesManager();
        private List<Keystroke> keystrokes = new List<Keystroke>();
        private List<KeystrokeEvent> keystrokeEventsBuffer;
        private KeystrokeStateController controller;
        private short[] uniqueKeyCount = new short[FileHelper.GetEnumCount<KeysList>()];
        KeyboardData keyboardData = new KeyboardData();
        private KeystrokesManager()
        {
            controller = KeystrokeStateController.GetStateController();
            keystrokeEventsBuffer = new List<KeystrokeEvent>();
        }
```

```
        public static KeystrokesManager GetKeyStrokesManager()
        {
            return _instance;
        }
        public static IntPtr CallbackFunction(Int32 code, IntPtr wParam, IntPtr lParam)
        {
            WM eventType = (WM)wParam;
            if (code >= 0 && (eventType == WM.KEYDOWN || eventType == WM.KEYUP || eventType == WM.SYSKEYDOWN ||
eventType == WM.SYSKEYUP))
            {
                //Task.Run(async() => { await Task.Run(() => {
AppManager.GetAppManager().CheckIfSessionChanged(); }); });
                var logMngr = GetKeyStrokesManager();
                if (AppManager.GetAppManager().IsBusy == true)
                {
                    return NativeMethods.CallNextHookEx(IntPtr.Zero, code, wParam, lParam);
                }
                AppManager.GetAppManager().CheckIfSessionChanged();
                KBDLLHOOKSTRUCT keyData = (KBDLLHOOKSTRUCT)Marshal.PtrToStructure(lParam,
typeof(KBDLLHOOKSTRUCT));
                var vKey = keyData.vkCode;
                KeystrokeEvent keystrokeEvent = new KeystrokeEvent();
                keystrokeEvent.EventTime = keyData.time;
                Keys defaultKeysEnum = (Keys)vKey;
                var key = KeyMapper.GetKeyEnum(defaultKeysEnum);
                if (key == KeysList.NoKey)
                {
                    return NativeMethods.CallNextHookEx(IntPtr.Zero, code, wParam, lParam);
                }
                keystrokeEvent.Key.Data = key;
                if (eventType == WM.KEYDOWN || eventType == WM.SYSKEYDOWN)
                {
                    keystrokeEvent.Type = KeystrokeType.KeyDown;
                    //Console.WriteLine("Down");
                }
                else if (eventType == WM.KEYUP || eventType == WM.SYSKEYUP)
                {
                    TimeManager.GetTimeManager().CreateNewAction();
                    keystrokeEvent.Type = KeystrokeType.KeyUp;
                    //Console.WriteLine("Up");
                }
                logMngr.InsertKeystrokeEvent(keystrokeEvent);
                //Console.WriteLine($"{btnStatus},{keyData.time},{defaultKeysEnum.GetDescription()},
{key.GetDescription()}");
            }
            return NativeMethods.CallNextHookEx(IntPtr.Zero, code, wParam, lParam);
        }
        private void InsertKeystrokeEvent(KeystrokeEvent key)
        {
            keystrokeEventsBuffer.Add(key);
        }
        public void SessionChanged()
        {
            uniqueKeyCount = new short[FileHelper.GetEnumCount<KeysList>()];
            keystrokes.Clear();
            keyboardData = new KeyboardData();
        }
        public KeyboardData GetKeyboardData()
        {
            KeystrokeMaker();
            keyboardData.StrokesCount = keystrokes.Count;
            keyboardData.BackspaceStrokesCount = uniqueKeyCount[(int)KeysList.Back];
            for (int i = 0; i < uniqueKeyCount.Length; i++)
            {
                if(uniqueKeyCount[i] > 0)
                {
                    keyboardData.UniqueKeysCount++;
                }
            }
            return keyboardData;
        }
        private void KeystrokeMaker()
        {
            for (int i = 0; i < keystrokeEventsBuffer.Count; i++)
            {
                if (keystrokeEventsBuffer[i] != null)
                {
                    if (keystrokeEventsBuffer[i].Type == KeystrokeType.KeyDown)
                    {
```

```
                        Keystroke keystroke = new Keystroke();
                        keystroke.Key = keystrokeEventsBuffer[i].Key;
                        keystroke.KeyDown = keystrokeEventsBuffer[i].EventTime;
                        uniqueKeyCount[keystroke.Key.KeyIndex]++;
                        for (int j = i + 1; j < keystrokeEventsBuffer.Count; j++)
                        {
                            if (keystrokeEventsBuffer[j] != null)
                            {
                                if (keystrokeEventsBuffer[j].Key.KeyIndex == keystroke.Key.KeyIndex)
                                {
                                    if (keystrokeEventsBuffer[j].Type == KeystrokeType.KeyUp)
                                    {
                                        keystroke.KeyUp = keystrokeEventsBuffer[j].EventTime;
                                        keyboardData.StrokeHoldTimes += keystroke.HoldTime;
                                        keystrokes.Add(keystroke);
                                        break;
                                    }
                                    else
                                    {
                                        keystrokeEventsBuffer[j] = null;
                                    }
                                }
                            }
                        }
                    }
                }
            }
            keystrokeEventsBuffer.Clear();
            //Console.WriteLine(keystrokes.Count);
        }
    }
}
namespace HRPMCore.Managers
{
    public class MouseManager
    {
        private static readonly MouseManager _instance = new MouseManager();
        private List<MouseClick> mouseClicks = new List<MouseClick>();
        private List<MouseClickEvent> mouseClickEventsBuffer;
        private MouseStateController controller;
        MouseData mouseData = new MouseData();
        private MouseManager()
        {
            controller = MouseStateController.GetStateController();
            mouseClickEventsBuffer = new List<MouseClickEvent>();
        }
        public static MouseManager GetMouseManager()
        {
            return _instance;
        }
        public static IntPtr CallbackFunction(Int32 code, IntPtr wParam, IntPtr lParam)
        {
            WM t = (WM)wParam;
            if (t == WM.MOUSEWHEEL || t == WM.RBUTTONDOWN || t == WM.RBUTTONUP || t == WM.LBUTTONDOWN || t ==
WM.LBUTTONUP || t == WM.XBUTTONUP || t == WM.XBUTTONDOWN || t == WM.MOUSEMOVE)
            {
                //Task.Run(async () => { await Task.Run(() => {
AppManager.GetAppManager().CheckIfSessionChanged(); }); });
                var mngr = GetMouseManager();
                if (AppManager.GetAppManager().IsBusy == true)
                {
                    return NativeMethods.CallNextHookEx(IntPtr.Zero, code, wParam, lParam);
                }
                AppManager.GetAppManager().CheckIfSessionChanged();
                MSLLHOOKSTRUCT butonData = (MSLLHOOKSTRUCT)Marshal.PtrToStructure(lParam,
typeof(MSLLHOOKSTRUCT));
                MouseClickEvent mouseClickEvent = new MouseClickEvent();
                mouseClickEvent.EventTime = butonData.time;
                if(t == WM.RBUTTONDOWN || t == WM.RBUTTONUP)
                {
                    mouseClickEvent.MouseButton.Data = MouseButtonList.RightButton;
                    if(t == WM.RBUTTONDOWN)
                    {
                        mouseClickEvent.Type = KeystrokeType.KeyDown;
                        //Console.WriteLine("Down");
                    }
                    else
                    {
                        TimeManager.GetTimeManager().CreateNewAction();
```

```
                    mouseClickEvent.Type = KeystrokeType.KeyUp;
                    //Console.WriteLine("Up");
                }
            }
            else if (t == WM.LBUTTONDOWN || t == WM.LBUTTONUP)
            {
                mouseClickEvent.MouseButton.Data = MouseButtonList.LeftButton;
                if (t == WM.LBUTTONDOWN)
                {
                    mouseClickEvent.Type = KeystrokeType.KeyDown;
                    //Console.WriteLine("Down");
                }
                else
                {
                    TimeManager.GetTimeManager().CreateNewAction();
                    mouseClickEvent.Type = KeystrokeType.KeyUp;
                    //Console.WriteLine("Up");
                }
            }
            else
            {
                return NativeMethods.CallNextHookEx(IntPtr.Zero, code, wParam, lParam);
            }
            mngr.InsertMouseClickEvent(mouseClickEvent);
        }
        return NativeMethods.CallNextHookEx(IntPtr.Zero, code, wParam, lParam);
    }
    private void InsertMouseClickEvent(MouseClickEvent mouseClickEvent)
    {
        mouseClickEventsBuffer.Add(mouseClickEvent);
    }
    public void SessionChanged()
    {
        mouseClicks.Clear();
        mouseData = new MouseData();
    }
    public MouseData GetMouseData()
    {
        MouseClickMaker();
        mouseData.MouseClickCount = (uint)mouseClicks.Count;
        return mouseData;
    }
    private void MouseClickMaker()
    {
        for (int i = 0; i < mouseClickEventsBuffer.Count; i++)
        {
            if (mouseClickEventsBuffer[i] != null)
            {
                if (mouseClickEventsBuffer[i].Type == KeystrokeType.KeyDown)
                {
                    MouseClick mouseClick = new MouseClick();
                    mouseClick.MouseButton = mouseClickEventsBuffer[i].MouseButton;
                    mouseClick.ButtonDown = mouseClickEventsBuffer[i].EventTime;
                    for (int j = i + 1; j < mouseClickEventsBuffer.Count; j++)
                    {
                        if (mouseClickEventsBuffer[j] != null)
                        {
                            if (mouseClickEventsBuffer[j].MouseButton.KeyIndex ==
mouseClick.MouseButton.KeyIndex)
                            {
                                if (mouseClickEventsBuffer[j].Type == KeystrokeType.KeyUp)
                                {
                                    mouseClick.ButtonUp = mouseClickEventsBuffer[j].EventTime;
                                    mouseData.MouseClickTotalTime += mouseClick.HoldTime;
                                    mouseClicks.Add(mouseClick);
                                    break;
                                }
                                else
                                {
                                    mouseClickEventsBuffer[j] = null;
                                }
                            }
                        }
                    }
                }
            }
        }
        mouseClickEventsBuffer.Clear();
    }
```

```
    }
}
namespace HRPMCore.Managers
{
    public class TimeManager
    {
        private static readonly TimeManager _instance = new TimeManager();
        DateTime lastSeen = DateTime.Now;
        double activeMinutes;
        double idleMinutes;
        Timer timer;
        DateTime startTime = DateTime.Now;
        public UsageTime Usage;
        private TimeManager()
        {
            timer = new Timer(60*60*1000);
            timer.Elapsed += Timer_Elapsed;
            timer.Start();
        }
        private void Timer_Elapsed(object sender, ElapsedEventArgs e)
        {
            CreateNewAction();
            Usage = new UsageTime();
            Usage.ActiveMinutes = activeMinutes;
            Usage.IdleMinutes = idleMinutes;
            Usage.StartTime = startTime;
            Usage.EndTime = DateTime.Now;
            activeMinutes = 0;
            idleMinutes = 0;
            startTime = DateTime.Now;
        }
        public static TimeManager GetTimeManager()
        {
            return _instance;
        }
        public void CreateNewAction()
        {
            if((DateTime.Now - lastSeen).TotalMinutes > 2)
            {
                idleMinutes += (DateTime.Now - lastSeen).TotalMinutes;
                lastSeen = DateTime.Now;
            }
            else
            {
                activeMinutes += (DateTime.Now - lastSeen).TotalMinutes;
                lastSeen = DateTime.Now;
            }
        }
    }
}
namespace HRPMCore.StateControllers
{
    public class KeystrokeStateController : StateController
    {
        private static readonly KeystrokeStateController _instance = new KeystrokeStateController();
        NativeMethods.HookProc callback = KeystrokesManager.CallbackFunction;
        private KeystrokeStateController()
        {
        }
        public static KeystrokeStateController GetStateController()
        {
            return _instance;
        }
        protected override void TerminateTask()
        {
            cts.Cancel();
            cts = new CancellationTokenSource();
            KeystrokesManager.GetKeyStrokesManager().GetKeyboardData();
        }
        protected override void RunTask(CancellationToken cancellationToken)
        {
            var module = Process.GetCurrentProcess().MainModule.ModuleName;
            var moduleHandle = NativeMethods.GetModuleHandle(module);
            hHook = NativeMethods.SetWindowsHookEx(HookType.WH_KEYBOARD_LL, callback, moduleHandle, 0);
            try
            {
                isRunning = true;
                while (true)
                {
```

```
                    NativeMethods.PeekMessage(IntPtr.Zero, IntPtr.Zero, 0x100, 0x109, 0);
                    Thread.Sleep(5);
                    cancellationToken.ThrowIfCancellationRequested();
                }
            }
            catch (Exception)
            {
                NativeMethods.UnhookWindowsHookEx(hHook);
                isRunning = false;
                return;
            }
        }
    }
}
namespace HRPMCore.StateControllers
{
    public class MouseStateController : StateController
    {
        private static readonly MouseStateController _instance = new MouseStateController();
        NativeMethods.HookProc callback = MouseManager.CallbackFunction;
        private MouseStateController()
        {
        }
        public static MouseStateController GetStateController()
        {
            return _instance;
        }
        protected override void TerminateTask()
        {
            cts.Cancel();
            cts = new CancellationTokenSource();
            KeystrokesManager.GetKeyStrokesManager().GetKeyboardData();
        }
        protected override void RunTask(CancellationToken cancellationToken)
        {
            var module = Process.GetCurrentProcess().MainModule.ModuleName;
            var moduleHandle = NativeMethods.GetModuleHandle(module);
            hHook = NativeMethods.SetWindowsHookEx(HookType.WH_MOUSE_LL, callback, moduleHandle, 0);
            try
            {
                isRunning = true;
                while (true)
                {
                    NativeMethods.PeekMessage(IntPtr.Zero, IntPtr.Zero, (uint)WM.MOUSEFIRST, (uint)WM.MOUSELAST,
0);
                    Thread.Sleep(5);
                    cancellationToken.ThrowIfCancellationRequested();
                }
            }
            catch (Exception)
            {
                NativeMethods.UnhookWindowsHookEx(hHook);
                isRunning = false;
                return;
            }
        }
    }
}
namespace HRPMCore.StateControllers
{
    public abstract class StateController : IStateController
    {
        public string filePath;
        protected CancellationTokenSource cts = new CancellationTokenSource();
        protected Task loggerTask;
        protected bool isRunning = false;
        protected IntPtr hHook;
        protected StateController()
        {
        }
        public bool IsRunning()
        {
            return isRunning;
        }
        public void Run()
        {
            if (loggerTask == null)
            {
                loggerTask = Task.Run(() => RunTask(cts.Token));
```

```
            }
            else
            {
                if (loggerTask.Status == TaskStatus.Running)
                {
                    return;
                }
                else if (loggerTask.Status == TaskStatus.Canceled || loggerTask.Status == TaskStatus.Faulted ||
loggerTask.Status == TaskStatus.RanToCompletion)
                {
                    loggerTask = Task.Run(() => RunTask(cts.Token));
                }
            }
        }
        public void Stop()
        {
            if (loggerTask == null)
            {
                return;
            }
            else
            {
                if (loggerTask.Status == TaskStatus.Running)
                {
                    TerminateTask();
                }
                else if (loggerTask.Status == TaskStatus.Canceled || loggerTask.Status == TaskStatus.Faulted)
                {
                    return;
                }
            }
        }
        protected abstract void TerminateTask();
        protected abstract void RunTask(CancellationToken cancellationToken);
    }
}
namespace HRPMCore
{
    public class StateControllersManager
    {
        private static readonly StateControllersManager _instance = new StateControllersManager();
        private AppManager mngr = AppManager.GetAppManager();
        List<IStateController> controllers = new List<IStateController>();
        private StateControllersManager()
        {
            controllers.Add(KeystrokeStateController.GetStateController());
            controllers.Add(MouseStateController.GetStateController());
        }
        public static StateControllersManager GetStateController()
        {
            return _instance;
        }
        public void Run()
        {
            foreach (var controller in controllers)
            {
                controller.Run();
            }
        }
        public void Stop()
        {
            foreach (var controller in controllers)
            {
                controller.Stop();
            }
            mngr.SaveSessionData(mngr.GetLastSession());
        }
        public List<AppSession> GetSessions()
        {
            mngr.SaveSessionData(mngr.GetLastSession());
            return mngr.GetAllSessions();
        }
        public void Initilize(string path)
        {
            mngr.Initialize(path);
        }
        public void ClearData()
        {
            mngr.ClearSessions();
```

```
        }
    }
}
namespace HRPMonitor.ViewModels
{
    public class MainControlViewModel : Screen, IHandle<PauseOptionModel>, ITasksWindowCaller
    {
        //private string _username = GlobalConfig.CurruntUser;
        private bool _isLogging = false;
        private string _startTimeLabel = UIStrings.StartTimeLabel;
        private string _stopTimeLabel = UIStrings.StopTimeLabel;
        private string _startTime;
        private string _stopTime;
        private LoggingStatus _loggingStatus = LoggingStatus.Stopped;
        private bool _isControlable = false;
        private bool _shouldSkip = false;
        private Timer appTimer;
        private DateTime startTime = new DateTime(2019, 10, 11, 0, 0, 0);
        private DateTime endTime = new DateTime(2019, 10, 11, 23, 50, 0);
        private double selectedPausePeriodTime = 0;
        private int pauseCountDown = 0;
        private bool isPaused = false;
        private bool isConnected = false;
        private readonly IEventAggregator _eventAggregator;
        private StateControllersManager stateMngr;
        private LogsManager logMngr;
        private FilesManager filelMngr;
        private User _user;
        private WorkTask curruntTask;
        private List<WorkTask> workTasks = new List<WorkTask>();
        #region UI Properties
        public string Username
        {
            get { return _user.UserName; }
            set { _user.UserName = value; }
        }
        public bool IsLogging
        {
            get { return _isLogging; }
            set
            {
                _isLogging = value;
                NotifyOfPropertyChange(() => IsLogging);
            }
        }
        public string StartTimeLabel
        {
            get { return _startTimeLabel; }
            set
            {
                _startTimeLabel = value;
                NotifyOfPropertyChange(() => StartTimeLabel);
            }
        }
        public string StopTimeLabel
        {
            get { return _stopTimeLabel; }
            set
            {
                _stopTimeLabel = value;
                NotifyOfPropertyChange(() => StopTimeLabel);
            }
        }
        public string StartTime
        {
            get { return _startTime; }
            set
            {
                _startTime = value;
                NotifyOfPropertyChange(() => StartTime);
            }
        }
        public string StopTime
        {
            get { return _stopTime; }
            set
            {
                _stopTime = value;
                NotifyOfPropertyChange(() => StopTime);
```

```
    }
}
public string ActionToolTip
{
    get
    {
        if (LoggingStatus == LoggingStatus.Stopped || LoggingStatus == LoggingStatus.Paused)
        {
            return UIStrings.RunActionToolTip;
        }
        else
        {
            return UIStrings.PauseActionToolTip;
        }
    }
}
public string StatusDescription
{
    get
    {
        if (LoggingStatus == LoggingStatus.Running)
        {
            return UIStrings.RunningStatusDescription;
        }
        else if (LoggingStatus == LoggingStatus.Paused)
        {
            return string.Format(UIStrings.PauesdStatusDescription, selectedPausePeriodTime);
        }
        else
        {
            return UIStrings.StoppedStatusDescription;
        }
    }
}
public bool IsControlable
{
    get { return _isControlable; }
    set
    {
        _isControlable = value;
        NotifyOfPropertyChange(() => IsControlable);
    }
}
public bool IsRunning
{
    get
    {
        if (LoggingStatus == LoggingStatus.Running)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
public bool IsStopped
{
    get
    {
        if (LoggingStatus == LoggingStatus.Stopped)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
public bool IsPaused
{
    get
    {
        if (LoggingStatus == LoggingStatus.Paused)
        {
            return true;
        }
```

```
        else
        {
            return false;
        }
    }
}
public string LoggingStatusText
{
    get
    {
        if (LoggingStatus == LoggingStatus.Running)
        {
            return UIStrings.RunningLoggingStatusText.ToUpper();
        }
        else if (LoggingStatus == LoggingStatus.Paused)
        {
            return UIStrings.PausedLoggingStatusText.ToUpper();
        }
        else
        {
            return UIStrings.StoppedLoggingStatusText.ToUpper();
        }
    }
}
public LoggingStatus LoggingStatus
{
    get { return _loggingStatus; }
    set
    {
        _loggingStatus = value;
        NotifyOfPropertyChange(() => LoggingStatus);
        NotifyOfPropertyChange(() => IsRunning);
        NotifyOfPropertyChange(() => IsStopped);
        NotifyOfPropertyChange(() => IsPaused);
        NotifyOfPropertyChange(() => LoggingStatusText);
        NotifyOfPropertyChange(() => ActionToolTip);
        NotifyOfPropertyChange(() => StatusDescription);
        ShowBalloon();
        SetLoggerStatus();
        PublishStatusCahngedEvent();
    }
}
public string TaskText
{
    get
    {
        return "Tasks";
    }
}
#endregion
/// <summary>
///     The Constructor
/// </summary>
public MainControlViewModel(IEventAggregator eventAggregator, User user)
{
    _user = user;
    stateMngr = StateControllersManager.GetStateController();
    logMngr = LogsManager.GetLogsManager();
    filelMngr = FilesManager.GetFilesManager();
    StartTime = startTime.ToString("HH:mm");
    StopTime = endTime.ToString("HH:mm");
    _eventAggregator = eventAggregator;
    _eventAggregator.Subscribe(this);
    appTimer = new Timer(1000);
    appTimer.Elapsed += AppTimer_Elapsed;
    appTimer.Start();
}
public async Task Toggle()
{
    if (LoggingStatus == LoggingStatus.Paused)
    {
        isPaused = false;
        StartTime = startTime.ToString("HH:mm");
        StartTimeLabel = UIStrings.StartTimeLabel;
        LoggingStatus = LoggingStatus.Running;
        logMngr.AddLog(new Log
        {
            Severity = LogSeverity.Medium,
            Type = LogType.LoggerStarted,
```

```
            Text = "Logger Started Manually."
        });
    }
    else if (LoggingStatus == LoggingStatus.Running)
    {
        await DialogHost.Show(new PauseDialogView(_eventAggregator), "RootDialog");
    }
}
public void OpenTasks()
{
    TasksWindow win = new TasksWindow(curruntTask, workTasks, this);
    win.ShowDialog();
}
public void Handle(PauseOptionModel message)
{
    selectedPausePeriodTime = message.SelectedPausePeriodTime;
    StartPauseTimer((int)message.SelectedPausePeriodTime);
}
private void StartPauseTimer(int periodInMinute)
{
    StartTimeLabel = UIStrings.PauseTimeLabel;
    pauseCountDown = periodInMinute * 60;
    isPaused = true;
    LoggingStatus = LoggingStatus.Paused;
    logMngr.AddLog(new Log
    {
        Severity = LogSeverity.High,
        Type = LogType.LoggerPaused,
        Text = $"Logger paused for {periodInMinute} mins."
    });
}
private void AppTimer_Elapsed(object sender, ElapsedEventArgs e)
{
    if (TimeManager.GetTimeManager().Usage != null)
    {
        //Console.WriteLine(TimeManager.GetTimeManager().Usage.IdleMinutes);
        var api = ApiHelper.GetApiHelper();
        api.PostUsageTime(TimeManager.GetTimeManager().Usage, _user);
        TimeManager.GetTimeManager().Usage = null;
    }
    TimeSpan nowtime = DateTime.Now.TimeOfDay;
    if (isConnected)
    {
        if (DateTime.Now.Second % 10 == 0)
        {
            if (logMngr.IsSync == false)
            {
                logMngr.SyncLogs(_user);
            }
        }
        if (DateTime.Now.Minute % 30  == 0)
        {
            if (!_shouldSkip)
            {
                _shouldSkip = true;
                filelMngr.CreateLocalFile(stateMngr.GetSessions());
                filelMngr.DeleteLiveDataFile();
                stateMngr.ClearData();
                if (filelMngr.IsSync == false)
                {
                    filelMngr.SyncFiles(_user);
                }
            }
        }
        else
        {
            _shouldSkip = false;
        }
    }
    if (isPaused)
    {
        pauseCountDown--;
        if (pauseCountDown == 0)
        {
            isPaused = false;
            //IsControlable = true;
            StartTime = UIStrings.StartTime;
            StartTimeLabel = UIStrings.StartTimeLabel;
            LoggingStatus = LoggingStatus.Running;
```

```
                    logMngr.AddLog(new Log
                    {
                        Severity = LogSeverity.Low,
                        Type = LogType.LoggerStarted,
                        Text = "Logger Started automatically after being paused."
                    });
                }
                else
                {
                    TimeSpan time = TimeSpan.FromSeconds(pauseCountDown);
                    StartTime = time.ToString(@"hh\:mm\:ss");
                }
            }
            if (LoggingStatus == LoggingStatus.Stopped &&
                nowtime > startTime.TimeOfDay && nowtime < endTime.TimeOfDay /*&& DateTime.Now.DayOfWeek !=
DayOfWeek.Saturday*/ && DateTime.Now.DayOfWeek != DayOfWeek.Sunday)
            {
                isConnected = true;
                LoggingStatus = LoggingStatus.Running;
                IsControlable = true;
                logMngr.AddLog(new Log
                {
                    Severity = LogSeverity.Low,
                    Type = LogType.LoggerStarted,
                    Text = "Logger Started automatically."
                });
                return;
            }
            if(
                (LoggingStatus == LoggingStatus.Running || LoggingStatus == LoggingStatus.Paused)
                &&
                (nowtime > endTime.TimeOfDay || nowtime < startTime.TimeOfDay))
            {
                LoggingStatus = LoggingStatus.Stopped;
                filelMngr.CreateLocalFile(stateMngr.GetSessions());
                filelMngr.DeleteLiveDataFile();
                stateMngr.ClearData();
                isConnected = false;
                IsControlable = false;
                logMngr.AddLog(new Log
                {
                    Severity = LogSeverity.Low,
                    Type = LogType.LoggerStopped,
                    Text = "Logger Stopped automatically."
                });
                return;
            }
        }
        private void ShowBalloon()
        {
            string title;
            string message;
            if(LoggingStatus == LoggingStatus.Running)
            {
                title = UIStrings.RunningSystemNotificationTitle;
                message = UIStrings.RunningSystemNotificationMessage;
            }
            else if(LoggingStatus == LoggingStatus.Paused)
            {
                title = UIStrings.PausedSystemNotificationTitle;
                message = UIStrings.PausedSystemNotificationMessage;
            } else
            {
                title = UIStrings.StoppedSystemNotificationTitle;
                message = UIStrings.StoppedSystemNotificationMessage;
            }
            BalloonMessageModel msg = new BalloonMessageModel { Message = message, Title = title };
            _eventAggregator.PublishOnUIThread(msg);
        }
        private void SetLoggerStatus()
        {
            if(LoggingStatus == LoggingStatus.Running)
            {
                stateMngr.Run();
            }
            else
            {
                stateMngr.Stop();
            }
```

```
        }
        private void PublishStatusCahngedEvent()
        {
            _eventAggregator.BeginPublishOnUIThread(new LoggingInfoModel
            {
                LoggingSatusText = LoggingStatusText,
                loggingStatus = LoggingStatus
            });
        }
        public void TaskAdded(WorkTask workTask)
        {
        }
        public void TaskCreated(WorkTask workTask)
        {
            curruntTask = workTask;
        }
        public async Task TaskSaved(WorkTask workTask)
        {
            var api = ApiHelper.GetApiHelper();
            await api.PostTask(workTask, _user);
        }
    }
}
namespace HRPMonitor.ViewModels
{
    public class ShellViewModel : Conductor<IScreen>, IHandle<LoggingInfoModel>, ILoginWindowCaller
    {
        private int _resizeBorder = 6;
        private int _outerMarginSize = 10;
        private int _windowRadius = 6;
        private int _titleHeight = 30;
        private WindowState _windowState = WindowState.Normal;
        private double _windowMinimumWidth = 400;
        private double _windowMinimumHeight = 400;
        private Visibility _windowVisibility = Visibility.Visible;
        private string _version = Assembly.GetExecutingAssembly().GetName().Version.ToString();
        private string _windowTitle = UIStrings.Title;
        private readonly IEventAggregator _eventAggregator;
        private LoggingStatus _loggingStatus = LoggingStatus.Stopped;
        private string _loggingStatusText;
        private User _user;
        Timer updateTime;
        /// <summary>
        ///     The Constructor
        /// </summary>
        public ShellViewModel(IEventAggregator eventAggregator, User user)
        {
            _eventAggregator = eventAggregator;
            _eventAggregator.Subscribe(this);
            if(user == null)
            {
                _user = user;
                WindowVisibility = Visibility.Collapsed;
                LoginWindow loginWin = new LoginWindow(this);
                loginWin.Show();
            }
            else
            {
                ActivateItem(new MainControlViewModel(_eventAggregator, user));
            }
        }
        public string Version
        {
            get { return "Version " + _version; }
            set { _version = value; }
        }
        public double WindowMinimumWidth
        {
            get { return _windowMinimumWidth; }
            set { _windowMinimumWidth = value; }
        }
        public double WindowMinimumHeight
        {
            get { return _windowMinimumHeight; }
            set { _windowMinimumHeight = value; }
        }
        public int TitleHeight
        {
            get { return _titleHeight; }
```

```
    set { _titleHeight = value; }
}
public string WindowTitle
{
    get { return _windowTitle; }
    set { _windowTitle = value; }
}
public Visibility WindowVisibility
{
    get { return _windowVisibility; }
    set
    {
        _windowVisibility = value;
        NotifyOfPropertyChange(() => WindowVisibility);
    }
}
public WindowState WindowState
{
    get { return _windowState; }
    set
    {
        _windowState = value;
        NotifyOfPropertyChange(() => WindowState);
        NotifyOfPropertyChange(() => ResizeBorderThickness);
        NotifyOfPropertyChange(() => OuterMarginSize);
        NotifyOfPropertyChange(() => OuterMarginSizeThickness);
        NotifyOfPropertyChange(() => WindowRadius);
        NotifyOfPropertyChange(() => WindowCornerRadius);
        NotifyOfPropertyChange(() => WindowCornerRadius);
    }
}
public int WindowRadius
{
    get
    {
        return WindowState == WindowState.Maximized ? 0 : _windowRadius;
    }
    set
    {
        _windowRadius = value;
    }
}
public int OuterMarginSize
{
    get
    {
        return WindowState == WindowState.Maximized ? 0 : _outerMarginSize;
    }
    set
    {
        _outerMarginSize = value;
    }
}
public int ResizeBorder
{
    get
    {
        return _resizeBorder;
    }
    set
    {
        _resizeBorder = value;
    }
}
public Thickness ResizeBorderThickness { get { return new Thickness(ResizeBorder + OuterMarginSize); } }
public CornerRadius WindowCornerRadius { get { return new CornerRadius(WindowRadius); } }
public Thickness OuterMarginSizeThickness { get { return new Thickness(OuterMarginSize); } }
public GridLength TitleHightGridLength { get { return new GridLength(TitleHeight + ResizeBorder); } }
public LoggingStatus LoggingStatus
{
    get { return _loggingStatus; }
    set
    {
        _loggingStatus = value;
        NotifyOfPropertyChange(() => LoggingStatus);
    }
}
public string LoggingStatusText
{
```

```
            get { return _loggingStatusText; }
            set
            {
                _loggingStatusText = value;
                NotifyOfPropertyChange(() => LoggingStatusText);
            }
        }
        public void Minimize()
        {
            WindowState = WindowState.Minimized;
        }
        public void Close()
        {
            WindowVisibility = Visibility.Hidden;
        }
        public void Show()
        {
            WindowVisibility = Visibility.Visible;
        }
        public void Restore()
        {
            WindowState = WindowState.Normal;
            WindowVisibility = Visibility.Visible;
        }
        public void Handle(LoggingInfoModel message)
        {
            LoggingStatusText = message.LoggingSatusText;
            LoggingStatus = message.loggingStatus;
        }
        protected override void OnDeactivate(bool close)
        {
            base.OnDeactivate(close);
            _eventAggregator.Unsubscribe(this);
        }
        public void OnLogin(User user)
        {
            _user = user;
            BinaryConnector.StaticSave(_user, GlobalConfig.UserDataFile);
            WindowVisibility = Visibility.Visible;
            ActivateItem(new MainControlViewModel(_eventAggregator, _user));
        }
    }
}
namespace HRPMonitor
{
    public class Bootstrapper : BootstrapperBase
    {
        private readonly SimpleContainer _container = new SimpleContainer();
        public User User { get; set; }
        public Bootstrapper()
        {
            CheckForMultipleProgramInstences();
            StateControllersManager.GetStateController().Initilize(GlobalConfig.LiveDataFilePath);
            CheckForUser();
#if !DEBUG
InitialSatrt();
#endif
            //CultureInfo.DefaultThreadCurrentCulture = new CultureInfo("tr-TR");
            //CultureInfo.DefaultThreadCurrentUICulture = new CultureInfo("tr-TR");
            Initialize();
        }
        private void CheckForUser()
        {
            try
            {
                User = BinaryConnector.StaticLoad<User>(GlobalConfig.UserDataFile);
            }
            catch
            {
                User = null;
            }
        }
        private void CheckForMultipleProgramInstences()
        {
            var processesWithTheSameName =
Process.GetProcessesByName(Path.GetFileNameWithoutExtension(Assembly.GetEntryAssembly().Location));
            if (processesWithTheSameName.Length > 1)
            {
                if (processesWithTheSameName.Length == 2)
```

```
                {
                    if (processesWithTheSameName[0].MainModule.FileName ==
processesWithTheSameName[1].MainModule.FileName)
                    {
                        File.AppendAllText(Path.Combine(Environment.GetFolderPath(
                         System.Environment.SpecialFolder.DesktopDirectory), "error.txt"), $"{DateTime.Now} -
error == 2");
                        Application.Current.Shutdown();
                    }
                }
                else
                {
                    File.AppendAllText(Path.Combine(Environment.GetFolderPath(
                         System.Environment.SpecialFolder.DesktopDirectory), "error.txt"), $"{DateTime.Now} -
error > 2");
                    Application.Current.Shutdown();
                }
            }
        }
        private void InitialSatrt()
        {
            using (var mgr = new UpdateManager("https://github.com/mhdb96/KDAnalyzer"))
            {
                SquirrelAwareApp.HandleEvents(
                  onInitialInstall: v =>
                  {
                      mgr.CreateShortcutForThisExe();
                      mgr.CreateRunAtWindowsStartupRegistry();
                  },
                  onAppUpdate: v => {
                      mgr.CreateShortcutForThisExe();
                      mgr.CreateRunAtWindowsStartupRegistry();
                  },
                  onAppUninstall: v =>
                  {
                      mgr.RemoveShortcutForThisExe();
                      mgr.RemoveRunAtWindowsStartupRegistry();
                  });
            }
        }
        protected override void OnStartup(object sender, StartupEventArgs e)
        {
            base.OnStartup(sender, e);
            DisplayRootViewFor<ShellViewModel>();
        }
        protected override void Configure()
        {
            _container.RegisterInstance(typeof(User), null, User);
            _container.Singleton<IWindowManager, WindowManager>();
            _container.Singleton<IEventAggregator, EventAggregator>();
            _container.RegisterPerRequest(typeof(ShellViewModel), null, typeof(ShellViewModel));
            _container.RegisterPerRequest(typeof(ShellView), null, typeof(ShellView));
        }
        protected override object GetInstance(Type serviceType, string key)
        {
            return _container.GetInstance(serviceType, key);
        }
        protected override IEnumerable<object> GetAllInstances(Type serviceType)
        {
            return _container.GetAllInstances(serviceType);
        }
        protected override void BuildUp(object instance)
        {
            _container.BuildUp(instance);
        }
    }
}
namespace HRPMSharedLibrary.DataAccess
{
    public class BinaryConnector
    {
        public static void StaticSave<T>(T obj, string path)
        {
            Directory.CreateDirectory(Path.GetDirectoryName(path));
            using (FileStream fs = new FileStream(path, FileMode.OpenOrCreate))
            {
                IFormatter formatter = new BinaryFormatter();
                formatter.Serialize(fs, obj);
            }
```

```
        }
        public static T StaticLoad<T>(string path)
        {
            T obj;
            Directory.CreateDirectory(Path.GetDirectoryName(path));
            using (FileStream fs = new FileStream(path, FileMode.Open))
            {
                IFormatter formatter = new BinaryFormatter();
                obj = (T)formatter.Deserialize(fs);
            }
            return obj;
        }
    }
}
namespace HRPMUILibrary.Helpers
{
    public class ApiHelper
    {
        private HttpClient apiClient;
        private static readonly ApiHelper _instance = new ApiHelper();
        private ApiHelper()
        {
            IntializeClient();
        }
        public static ApiHelper GetApiHelper()
        {
            return _instance;
        }
        private void IntializeClient()
        {
            apiClient = new HttpClient();
            apiClient.BaseAddress = new Uri("https://localhost:44360/");
            apiClient.DefaultRequestHeaders.Clear();
            apiClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
        }
        public async Task<bool> PostFile(BinaryFile file, User user)
        {
            file.User = user;
            using (HttpResponseMessage res = await apiClient.PostAsJsonAsync("api/files/UploadFile", file))
            {
                if (res.IsSuccessStatusCode)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }
        public async Task<bool> PostLog(Log log, User user)
        {
            log.User = user;
            using (HttpResponseMessage res = await apiClient.PostAsJsonAsync("api/logs/postlog", log))
            {
                if (res.IsSuccessStatusCode)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }
        public async Task<bool> GetServerStatus()
        {
            using (HttpResponseMessage res = await apiClient.GetAsync("api/checkers/isalive"))
            {
                if (res.IsSuccessStatusCode)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }
```

```
        public async Task<List<Department>> GetDepartments()
        {
            using (HttpResponseMessage res = await apiClient.GetAsync("api/departments/getall"))
            {
                if (res.IsSuccessStatusCode)
                {
                    return await res.Content.ReadAsAsync<List<Department>>();
                }
                else
                {
                    return null;
                }
            }
        }
        public async Task<List<User>> GetUsersByDepartment(int id)
        {
            using (HttpResponseMessage res = await apiClient.GetAsync($"api/users/getbydepid/{id}"))
            {
                if (res.IsSuccessStatusCode)
                {
                    return await res.Content.ReadAsAsync<List<User>>();
                }
                else
                {
                    return null;
                }
            }
        }
        public async Task<bool> GetUserPassword(User user)
        {
            using (HttpResponseMessage res = await apiClient.PostAsJsonAsync("api/users/checkpassword", user))
            {
                if (res.IsSuccessStatusCode)
                {
                    return await res.Content.ReadAsAsync<bool>();
                }
                else
                {
                    return false;
                }
            }
        }
        public async Task<bool> PostTask(WorkTask task, User user)
        {
            task.User = user;
            using (HttpResponseMessage res = await apiClient.PostAsJsonAsync("api/tasks/PostTask", task))
            {
                if (res.IsSuccessStatusCode)
                {
                    task = null;
                    return await res.Content.ReadAsAsync<bool>();
                }
                else
                {
                    return false;
                }
            }
        }
        public async Task<bool> PostUsageTime(UsageTime usage, User user)
        {
            usage.User = user;
            using (HttpResponseMessage res = await apiClient.PostAsJsonAsync("api/usagetimes/postusage", usage))
            {
                if (res.IsSuccessStatusCode)
                {
                    usage = null;
                    return await res.Content.ReadAsAsync<bool>();
                }
                else
                {
                    return false;
                }
            }
        }
    }
}
namespace HRPMUILibrary.Logic.Files
{
    public class FilesManager
```

```
{
    private static readonly FilesManager _instance = new FilesManager();
    private Queue<BinaryFile> files = new Queue<BinaryFile>();
    public bool IsSync
    {
        get
        {
            return files.Count <= 0;
        }
    }
    private FilesManager()
    {
        try
        {
            files = BinaryConnector.StaticLoad<Queue<BinaryFile>>(GlobalConfig.DataCacheFilePath);
        }
        catch (Exception e)
        {
            files = new Queue<BinaryFile>();
            LogsManager.LogToTextFile(new ErrorLog
            {
                Severity = LogSeverity.Low,
                Type = ErrorType.IOError,
                Text = "No cahced data file was found. - " + e.Message
            });
        }
    }
    public static FilesManager GetFilesManager()
    {
        return _instance;
    }
    public void AddFile(BinaryFile file)
    {
        files.Enqueue(file);
        LogsManager.LogToTextFile(new Log
        {
            Severity = LogSeverity.Low,
            Type = LogType.FileCreated,
            Text = "New data file created."
        });
    }
    public async Task SyncFiles(User user)
    {
        var api = ApiHelper.GetApiHelper();
        while (files.Count > 0)
        {
            bool isPosted;
            try
            {
                // !!!!!!!!!!!!! NOT SENDING DATA TO SERVER !!!!!!!!!!!!!
                isPosted = await api.PostFile(files.Peek(), user);
                if (isPosted)
                {
                    files.Dequeue();
                }
            }
            catch (Exception)
            {
                BinaryConnector.StaticSave(files, GlobalConfig.DataCacheFilePath);
                LogsManager.LogToTextFile(new ErrorLog
                {
                    Severity = LogSeverity.Medium,
                    Type = ErrorType.ConnectionError,
                    Text = "Failed to sync data files, no connection to the server."
                });
                break;
            }
        }
        if (files.Count == 0)
        {
            if (File.Exists(GlobalConfig.DataCacheFilePath))
            {
                File.Delete(GlobalConfig.DataCacheFilePath);
                LogsManager.LogToTextFile(new Log
                {
                    Severity = LogSeverity.Low,
                    Type = LogType.FileDeleted,
                    Text = "Data cache file deleted after syncing properly."
                });
```

```
                }
            }
        }
        public void CreateLocalFile(List<AppSession> data)
        {
            BinaryConnector.StaticSave(data, GlobalConfig.DataFilePath);
            BinaryFile file =  EncodeIntoBinaryFile(GlobalConfig.DataFilePath);
            AddFile(file);
        }
        private BinaryFile EncodeIntoBinaryFile(string path)
        {
            byte[] bytes = File.ReadAllBytes(path);
            BinaryFile file = new BinaryFile();
            file.Bytes = bytes;
            file.Name = Path.GetFileName(path);
            return file;
        }
        public void DeleteLiveDataFile()
        {
            if (File.Exists(GlobalConfig.LiveDataFilePath))
            {
                File.Delete(GlobalConfig.LiveDataFilePath);
                LogsManager.LogToTextFile(new Log
                {
                    Severity = LogSeverity.Low,
                    Type = LogType.FileDeleted,
                    Text = "Live data file deleted at the end of the work day."
                });
            }
        }
    }
}
namespace HRPMUILibrary.Logic.Logs
{
    public class LogsManager
    {
        private static readonly LogsManager _instance = new LogsManager();
        private Queue<Log> logs;
        public bool IsSync
        {
            get
            {
                return logs.Count <= 0;
            }
        }
        private LogsManager()
        {
            try
            {
                logs = BinaryConnector.StaticLoad<Queue<Log>>(GlobalConfig.LogCacheFilePath);
            }
            catch (Exception)
            {
                LogToTextFile(new ErrorLog
                {
                    Severity = LogSeverity.Low,
                    Type = ErrorType.IOError,
                    Text = "No cahced log file was found."
                });
                logs = new Queue<Log>();
            }
        }
        public static LogsManager GetLogsManager()
        {
            return _instance;
        }
        public void AddLog(Log log)
        {
            LogToTextFile(log);
            logs.Enqueue(log);
        }
        public async Task SyncLogs(User user)
        {
            var api = ApiHelper.GetApiHelper();
            while(logs.Count > 0)
            {
                bool isPosted;
                try
                {
```

```
                    // !!!!!!!!!!!!!! NOT SENDING DATA TO SERVER !!!!!!!!!!!!!!
                    isPosted = await api.PostLog(logs.Peek(), user);
                    if (isPosted)
                    {
                        logs.Dequeue();
                    }
                }
                catch (Exception)
                {
                    LogToTextFile(new ErrorLog
                    {
                        Severity = LogSeverity.Medium,
                        Type = ErrorType.ConnectionError,
                        Text = "Failed to sync logs, no connection to the server."
                    });
                    BinaryConnector.StaticSave(logs, GlobalConfig.LogCacheFilePath);
                    break;
                }
            }
            if(logs.Count == 0)
            {
                if (File.Exists(GlobalConfig.LogCacheFilePath))
                {
                    File.Delete(GlobalConfig.LogCacheFilePath);
                    LogToTextFile(new Log
                    {
                        Severity = LogSeverity.Low,
                        Type = LogType.FileDeleted,
                        Text = "Log cache file deleted after syncing properly."
                    });
                }
            }
        }
        public static void LogToTextFile(ILog log)
        {
            Directory.CreateDirectory(Path.GetDirectoryName(GlobalConfig.logFilePath));
            File.AppendAllText(GlobalConfig.logFilePath, log.GetLogText() + Environment.NewLine);
        }
    }
}
namespace HRPMUILibrary
{
    public static class GlobalConfig
    {
        private readonly static string _mainFolderName = "HRPM_Files";
        private readonly static string _dataFilesFolderName = "Data";
        private readonly static string _logFilesFolderName = "Logs";
        private readonly static string _cacheFilesFolderName = ".cache";
        private static string _dataFileName= "data-file_{0}.hdf";
        private static string _logFileName= "log-file_{0}.hlf";
        private readonly static string _logCacheFileName= "log-cache-file.hcf";
        private readonly static string _dataCacheFileName = "data-cache-file.hcf";
        private readonly static string _liveDataFileName = "live-data-file.hcf";
        private readonly static string _userDataFile = "user-data-file.user";
        private readonly static string ProjectRootFolder =
Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
        public static string CurruntUser
        {
            get
            {
                return System.Security.Principal.WindowsIdentity.GetCurrent().Name;
            }
        }
        public static string MainFolderPath
        {
            get { return Path.Combine(ProjectRootFolder, _mainFolderName); }
        }
        public static string DataFilesFolderPath
        {
            get { return Path.Combine(ProjectRootFolder, _mainFolderName, _dataFilesFolderName); }
        }
        public static string LogFilesFolderPath
        {
            get { return Path.Combine(ProjectRootFolder, _mainFolderName, _logFilesFolderName); }
        }
        public static string CacheFilesFolderPath
        {
            get { return Path.Combine(ProjectRootFolder, _mainFolderName, _cacheFilesFolderName); }
        }
```

```
        public static string DataFilePath
        {
            get
            {
                return Path.Combine(ProjectRootFolder, _mainFolderName, _dataFilesFolderName,
DateTime.Now.ToString("dd-MM-yyyy"), string.Format(_dataFileName, DateTime.Now.ToString("HH-mm")));
            }
        }
        public static string logFilePath
        {
            get
            {
                return Path.Combine(ProjectRootFolder, _mainFolderName, _logFilesFolderName, string.Format(
_logFileName, DateTime.Now.ToString("dd-MM-yyyy")));
            }
        }
        public static string LogCacheFilePath
        {
            get { return Path.Combine(ProjectRootFolder, _mainFolderName, _cacheFilesFolderName,
_logCacheFileName); }
        }
        public static string DataCacheFilePath
        {
            get { return Path.Combine(ProjectRootFolder, _mainFolderName, _cacheFilesFolderName,
_dataCacheFileName); }
        }
        public static string LiveDataFilePath
        {
            get { return Path.Combine(ProjectRootFolder, _mainFolderName, _cacheFilesFolderName,
_liveDataFileName); }
        }
        public static string UserDataFile
        {
            get { return Path.Combine(ProjectRootFolder, _mainFolderName, _userDataFile); }
        }
    }
}
namespace HRPMWebAPI.Controllers
{
    public class DepartmentsController : ApiController
    {
        [Route("api/departments/getall")]
        [HttpGet]
        public List<Department> GetDepartments()
        {
            var dbDeps = GlobalConfig.Connection.Department_SelectAll();
            List<Department> localDeps = new List<Department>();
            foreach (var dep in dbDeps)
            {
                localDeps.Add(new Department { Id = dep.Id, Name = dep.Name });
            }
            return localDeps;
        }
    }
}
namespace HRPMWebAPI.Controllers
{
    public class FilesController : ApiController
    {
        [Route("api/files/UploadFile")]
        [HttpPost]
        public void UploadFile(BinaryFile file)
        {
            DataFileModel model = new DataFileModel();
            model.Date = file.Date;
            model.User.Id = file.User.Id;
            model.Path = FileManager.GetFileManager().SaveSessionsFile(file);
            GlobalConfig.Connection.File_Insert(model);
            var sessions = BinaryConnector.StaticLoad<List<AppSession>>(model.Path);
            foreach (var session in sessions)
            {
                if (session.MouseData != null && session.EndTime != DateTime.MinValue)
                {
                    AppModel appModel = new AppModel();
                    appModel.ProcessName = session.App.ProcessName;
                    GlobalConfig.Connection.App_Insert(appModel);
                    SessionModel sessionModel = new SessionModel();
                    sessionModel.AppId = appModel.Id;
                    sessionModel.BackspaceStrokesCount = session.KeyboardData.BackspaceStrokesCount;
```

```
                    sessionModel.StrokesCount = session.KeyboardData.StrokesCount;
                    sessionModel.StrokeHoldTimes = session.KeyboardData.StrokeHoldTimes;
                    sessionModel.UniqueKeysCount = session.KeyboardData.UniqueKeysCount;
                    sessionModel.MouseClickCount = (int)session.MouseData.MouseClickCount;
                    sessionModel.MouseClickTotalTime = (int)session.MouseData.MouseClickTotalTime;
                    sessionModel.StartTime = session.StartTime;
                    sessionModel.EndTime = session.EndTime;
                    sessionModel.UserId = model.User.Id;
                    sessionModel.AppId = appModel.Id;
                    GlobalConfig.Connection.Session_Insert(sessionModel);
                    if (session.App.Type == HRPMSharedLibrary.Enums.AppType.Browser)
                    {
                        DomainModel domainModel = new DomainModel();
                        domainModel.Domain = session.App.Content;
                        GlobalConfig.Connection.Domain_Insert(domainModel);
                        GlobalConfig.Connection.SessionDomain_Insert(sessionModel.Id, domainModel.Id);
                    }
                }
            }
        }
    }
}
namespace HRPMWebAPI.Controllers
{
    public class LogsController : ApiController
    {
        [Route("api/logs/PostLog")]
        [HttpPost]
        public void PostLog(Log log)
        {
            LogModel model = new LogModel();
            model.Text = log.Text;
            model.Type = log.Type;
            model.Severity = log.Severity;
            model.User.Username = log.User.UserName;
            model.User.Id = log.User.Id;
            model.Date = log.Date;
            GlobalConfig.Connection.Log_Insert(model);
        }
    }
}
namespace HRPMWebAPI.Controllers
{
    public class TasksController : ApiController
    {
        [Route("api/tasks/PostTask")]
        [HttpPost]
        public void PostTask(WorkTask task)
        {
            TaskModel model = new TaskModel();
            model.UserId = task.User.Id;
            model.Title = task.Title;
            model.Description = task.Description;
            model.StartTime = task.StartTime;
            model.EndTime = task.EndTime;
            GlobalConfig.Connection.Task_Insert(model);
        }
    }
}
namespace HRPMWebAPI.Controllers
{
    public class UsageTimesController : ApiController
    {
        [Route("api/usagetimes/postusage")]
        [HttpPost]
        public void PostUsageTime(UsageTime usage)
        {
            UsageTimeModel model = new UsageTimeModel();
            model.UserId = usage.User.Id;
            model.ActiveMinutes = usage.ActiveMinutes;
            model.IdleMinutes = usage.IdleMinutes;
            model.StartTime = usage.StartTime;
            model.EndTime = usage.EndTime;
            GlobalConfig.Connection.UsageTime_Insert(model);
        }
    }
}
namespace HRPMWebAPI.Controllers
{
```

```
    public class UsersController : ApiController
    {
        [Route("api/users/getbydepid/{id}")]
        [HttpGet]
        public List<User> GetUsersByDepartmentId(int id)
        {
            var dbUsers = GlobalConfig.Connection.User_GetByDepartment(id);
            List<User> localUsers = new List<User>();
            foreach (var user in dbUsers)
            {
                localUsers.Add(new User { Id = user.Id, UserName = user.Username });
            }
            return localUsers;
        }
        [Route("api/users/checkpassword")]
        [HttpPost]
        public bool CheckPasswordForUser(User user)
        {
            string userPassword = GlobalConfig.Connection.User_GetPassword(user.Id);
            if(userPassword == user.Password)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
    }
}
```